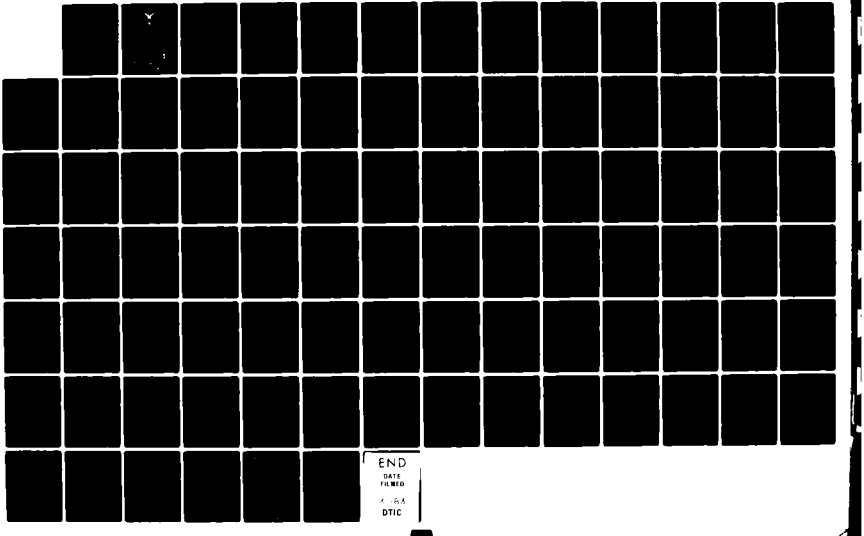
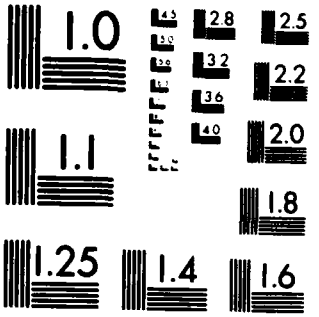
AD-A124 692 GRID-BASED LINE DRAWING QUANTIZATION(U) AIR FORCE INST 1/1
OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING
K R JONES DEC 82 AFIT/GE/EE/82D-41

UNCLASSIFIED F/G 12/1 NL



END
DATE FILMED
DTIC



AD A124692

AFIT/GE/EE/82D-41

GRID-BASED LINE DRAWING QUANTIZATION

THESIS

AFIT/GE/EE/82D-41

Keith R. Jones
2nd Lt. USAF

AFIT/GE/EE/82D-41

GRID-BASED LINE DRAWING QUANTIZATION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Keith R. Jones, B.S.

2nd Lt. USAF

Graduate Electrical Engineering

December 1982

Acknowledgements

I would like to thank my advisor, Major Ken Castor of the Air Force Institute of Technology, for proposing this thesis topic. I am also grateful for his assistance and encouragement throughout the project. I would also like to thank my family and friends for their constant support and encouragement.

Contents

List of Figures

Abstract

This paper documents a quantitative analysis of the performance of the generalized chain codes when used to quantize specific periodic waveforms. The analysis was performed using a software simulation of the various forms of the code to encode sine waves and circular waves. The performance of the coding schemes was measured in terms of the encoding rate and the area error in the quantization.

Comparisons were made of the performance of codes for a particular function when the initial phase was altered. Comparisons were also made of the performance of the different forms of the codes used in quantizing the same function. Finally, the performance of the codes for the sine wave was compared with the performance of the same code for a circular wave having the same amplitude and period.

I. Introduction

In many engineering and Air Force applications the ability to quantize, process, and store two-dimensional image data efficiently is very important. The general method for digitizing such image information is two-dimensional sampling. In order to obtain significant resolution the process requires a large memory space and lengthy processing time. For some images, two-dimensional sampling is not an efficient digitization scheme.

One example of such an image is the line drawing. A line drawing is an image which consists entirely of thin lines on a contrasting background. Contour maps, graphs, and printed texts are examples of line drawings. A family of quantization and encoding schemes has been developed which provides a more efficient method for digitizing line drawing images. These schemes are collectively called generalized chain codes. At the present time, the theory for analyzing the performance of the generalized chain codes is limited to primarily qualitative performance measures. There exists no general theoretical means for predicting the performance of the codes in a quantitative manner.

The objective of this thesis was to develop a quantitative comparison of the performance of various forms of the generalized chain codes when used to quantize certain periodic waveforms. The periodic waveforms used in the evaluation are sine waves and circular waves. The sine wave was chosen because it represents a line drawing with a continuously changing radius of curvature. The circular wave represents a line drawing with

a constant radius of curvature. The analysis was accomplished using a computer simulation of the encoding algorithms to perform the quantization. The quantized image was then compared to the original waveform and appropriate performance measures were computed.

The organization of the material presented in this thesis parallels the approach to the problem. In Chapter II, the mechanics of the generalized chain codes are discussed. Then in Chapter III, the algorithm for performing the quantization and for analyzing the performance is given. This is followed in Chapter IV by a description of the software which implements the algorithm. In the final chapter, the results are presented and discussed and recommendations for future studies are made.

II. The Generalized Chain Code

The generalized chain codes comprise a family of methods for quantizing and encoding line drawing images. The quantization of the image is performed by superimposing a grid of some specified mesh size onto the line drawing and selecting a set of the nodes of the grid to represent the image. The nodes of the grid are defined as the intersections of the horizontal and vertical grid lines. This set of nodes is then encoded using a method by which each node is represented by a binary number which indicates that node's position relative to the preceding node. The quantized and encoded version of the line drawing is, therefore, a sequence of binary numbers. This sequence is called a chain.

This procedure describes the basic structure of all of the generalized chain codes. The specific forms of the code differ in the rules which govern the selection of the grid nodes which represent the line drawing and the binary numbers which are used to encode the nodes. These quantization rules and encoding procedures will now be discussed in greater detail. The simplest form of the generalized chain codes, the grid intersect code, will be introduced first. This simple form will then be extended in a discussion of the more complex codes. The chapter will be concluded with a discussion of a set of qualitative performance criteria.

The Grid Intersect Code

The initial step in the quantization procedure for the grid intersect code is to trace the path of the line drawing from some assumed starting point to the first point where the line drawing intersects a grid line. This first grid line intersect may be located at the starting point if the starting point occurs on a grid line. The grid node closest to this intersection is initialized as the first node of the quantized set. The tracing of the path of the line drawing is continued and each point where a grid intersect occurs is determined in sequence. As each intersection is encountered, the nearest grid node is identified. In this manner, a grid node is selected for each grid intersect; however, if two or more successive intersections correspond to the same grid node, that grid node is selected only once.

For any particular node of the quantized set, the next node selected must be one of the eight adjacent nodes, as indicated in Figure 1-1.

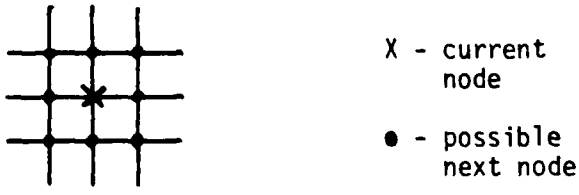


Figure 2-1 Possible Next Nodes of Ring 1

These eight nodes form a square ring which is called ring 1. Using this ring, a variation of the grid intersect code can be suggested. This variation, which I will call the (1)-code, consists of locating the first grid intersect on ring 1 and selecting the grid node closest to it as the

next node of the quantized set. The difference in the grid intersect code and the (1)-code is illustrated in Figure 2-2. In this example, node A is

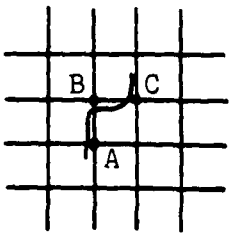


Figure 2-2 Example Illustrating the Grid Intersect Code and the (1)-Code

the current node. If the quantization is performed using the grid intersect code, node B and then node C are selected. If the quantization is performed using the (1)-code, however, only the node C is selected. Thus, there are two possible quantization schemes for selecting ring 1 next nodes, the grid intersect code and the (1)-code. The former provides better performance in terms of the quantization error, while the latter provides better performance in terms of the quantization rate (that is, the number of nodes in the quantized set).

For either of these quantization methods, the encoding procedure is the same. An octal integer (or three-bit binary number) is associated with each of the nodes in the ring as shown in Figure 2-3. This assignment

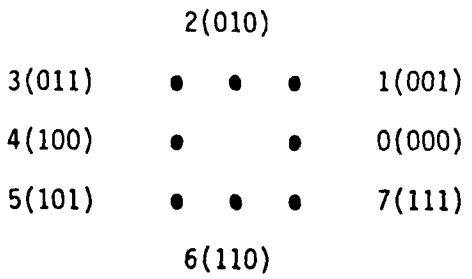


Figure 2-3 Encoding Assignment for Ring 1

is arbitrary, but must remain consistent in the encoding and decoding process. The grid nodes of the quantization are encoded by representing the relative position between nodes by one of these eight octal integers. For example, the line drawing of Figure 2-4 is represented by the chain, 120070.

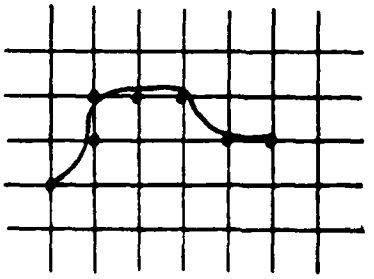


Figure 2-4 Example of Encoding Using the Grid Intersect Code

Higher Order Codes

These quantization and encoding procedures can easily be extended to higher order codes. The set of possible next nodes is allowed to include nodes from rings other than ring 1. The ring is defined to be that square ring for which the distance from the current node to the ring along either axis is equal to one times the mesh size of the grid. A general chain code can include any number and combination of rings. A code can be identified by the rings it contains. For example, the (1,2)-code includes ring 1 and ring 2.

The quantization procedure for a general chain code is as follows. The first intersection between the largest ring of the code and the line drawing is found and the grid node nearest that intersection is identified. It now must be determined whether this node accurately represents this portion of the line drawing. The process for doing so, which I will

call node validation, is performed by examining the grid intersects between the current node and the ring to determine if they are within a specified area of precision. This area of precision is defined as the area enclosed by a line segment from the current node to the point one-half the grid size clockwise on the ring from the possible next node, a similar line segment to the point one-half the grid size counter-clockwise on the ring from the possible next node, and the ring. The area of precision for a node on ring 3 is illustrated by the shaded area of Figure 2-5. If all of the grid intersects occur within this area, the

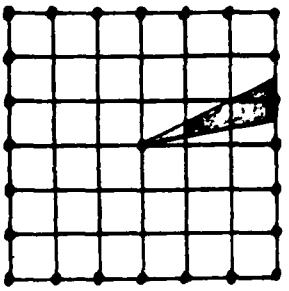


Figure 2-5 Area of Precision for a Ring 3 Node

node becomes part of the quantization. Otherwise, the process of selecting and validating a node is repeated for the next smaller ring of the code.

This process is repeated until a node is validated or the smallest ring of the code is reached. In the latter case, there are two alternatives for selecting the next node analogous to the choice between the grid intersect code and the (1)-code (the only difference is that, in this situation, the ring is not necessarily ring 1). The first alternative is to find the first grid intersect for which the nearest grid node

is on the ring. The node is then added to the quantization. The other alternative is to find the intersect between the line drawing and the ring and to add the node closest to that intersection to the quantization. In this thesis the latter method is used.

The validation procedure is illustrated in Figure 2-6 for a (1,3)-code. In Figure 2-6(a), A represents the current node of the quantized

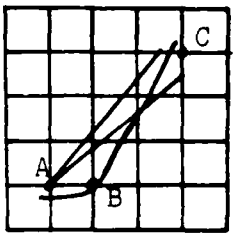


Figure 2-6(a) Example of Node Validation Failure

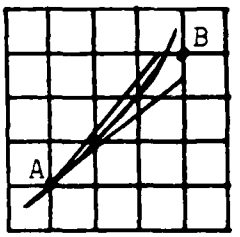


Figure 2-6(b) Example of Successful Node Validation

set and C represents the possible next node on ring 3. In this case, the node validation failed and the possible next node on ring 1 was formed. Since ring 1 is the smallest ring of the code, the node on this ring, represented by B, would become the next node of the quantized set. Figure 2-6(b) represents the case where the node on ring 3, indicated by B, was validated. This node would then become the next node of the quantized set.

The encoding procedure for the higher order codes is much the same as for the grid intersect code. A binary number is associated with each node on each ring of the code. The binary number for each node of the quantization, corresponding to the relative position between the nodes, form a chain which is the quantized and encoded representation of the

line drawing. There is no standard convention governing the assignment of numbers to the nodes.

Evaluation of a General Chain Code

In evaluating the usefulness of a general chain code, five criteria for comparison have been suggested [3]: (1) compactness, (2) precision, (3) smoothness, (4) ease of encoding and decoding, and (5) facility for processing. The relative importance of each of these criteria is dependent on the intended application. In the following discussion, each of the criteria will be reviewed in relation to the various aspects of the general chain code.

Compactness is of importance if the primary purpose of the encoding is storage or transmission. The lower order codes which have fewer possible next nodes require fewer bits for encoding each node. The higher order code, utilizing the larger rings, may require fewer nodes to represent the line drawing. The curvature characteristics (max, min, rate of change) of the line drawing is very important in the evaluation of the trade-offs between the lower- and higher-order codes for this application.

The precision with which the encoded data represents the line drawing is important when the quantitative aspects of the encoding are of primary concern. The precision is highly dependent on the grid size. The codes utilizing the smaller ring may be more accurate where the radius of curvature is small. In applications where the encoded data is to be displayed, the smoothness of the approximation may be significant. For this case, the codes utilizing the larger rings provide greater angular resolution.

The ease of encoding and decoding is important when large quantities of data are involved. The lower order codes are simpler and, therefore, easier to encode and decode; but the advantage is slight when performed by a digital computer. When an extensive amount of processing is to be performed, the simplicity of processing more strongly affects the choice of the code. The higher order codes, which generate fewer nodes, may be advantageous in this case since fewer nodes require less processing time.

Summary

In this chapter, the basic structure of the generalized chain codes was introduced. The quantization and encoding procedures for the codes were discussed and a set of criteria for evaluating the various aspects of the codes was presented.

III. The Analysis

The purpose of this chapter is to describe the analysis which is used in comparing the performance of various forms of the generalized chain codes. The analysis includes the quantization of a line drawing and the calculation of the performance measures. The simulation of the encoding procedure is not necessary since the pertinent information concerning the encoding can be derived from the quantization and the code parameters. The quantization is performed in two steps. The first step is to calculate the grid intersects and the second step is to determine the nodes of the quantization. Thus, the analysis can be divided into three separate steps:

1. Calculation of the grid intersects
2. Calculation of the nodes of the quantization
3. Calculation of the performance

Each of these steps will now be discussed in detail.

Calculation of the Grid Intersects

The algorithm for calculating the grid intersects is based on the following three assumptions.

1. The grid is located in the right half plane of a cartesian coordinate system such that the vertical grid lines are parallel to the y-axis.
2. The line drawing can be defined by a function which is at least piece-wise continuous and invertible, and whose domain extends from zero to some finite positive number. (Note:

The function does not have to be invertible in the strict mathematical sense, but its inverse must be implementable in the software.)

3. The grid size is small enough such that the derivative of the function defining the line drawing has at most one zero between any two vertical grid lines.

The calculation of the grid intersects is accomplished by a process in which the domain of the function is, in effect, divided into intervals defined by the vertical grid lines as illustrated in Figure 3-1. A procedure for calculating the grid intersects in order in an interval is then

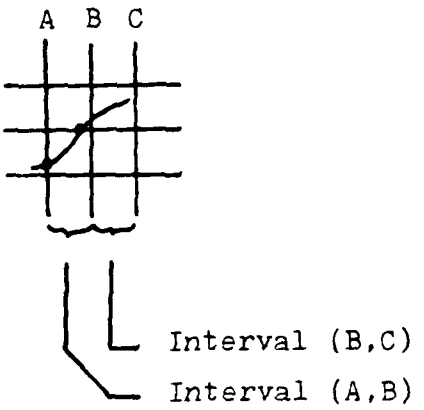


Figure 3-1 Finding Grid Intersects by Partitioning the Domain of the Function into Intervals

repeated for each interval in succession. As a result, each grid intersect is calculated in the order in which they would be encountered if the path of the line drawing was traced from beginning to end. A grid intersect is identified by the coordinates of the point where the function crosses a horizontal or vertical grid line. One coordinate is known from

the coordinate of the grid line. The calculation of a grid intersect, therefore, is the evaluation of the function or its inverse to determine the second coordinate.

The procedure for calculating the grid intersect in an arbitrary interval is as follows. The grid intersect on the vertical grid line at the beginning of the interval is first calculated. A check is then made to determine if the point is the end of the domain of the function, in which case, the calculation of grid intersects is completed. Otherwise, a check is made to determine if the vertical grid line at the end of the interval exceeds the end of the domain. In this case, the interval is shortened to correspond to the end of the domain. The grid intersects within the interval are then calculated.

The calculation of the grid intersects within the interval is performed by a procedure for which the function must be strictly non-increasing or non-decreasing in the interval. In order to satisfy this constraint, the sign of the derivative at the lower bound of the interval is compared with that at the upper bound. If these signs are opposite, the interval is divided into subintervals separated by the point where the derivative is zero. This is illustrated in Figure 3-2. The grid intersects

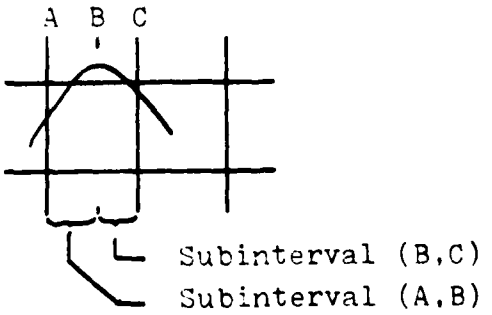


Figure 3-2 Dividing Intervals into Subintervals

within an interval satisfying the constraint are then calculated by first identifying the first and last horizontal grid line within the range of the function in the interval (excluding the possible horizontal grid line at the upper or lower bound) and then calculating the grid intersect on each of these grid lines in order. If the interval was subdivided, then, immediately following the calculations for the first subinterval, a check is made to determine if the point where the interval was subdivided intersects a horizontal grid line. If the interval was shortened due to the occurrence of the end of the domain of the function, then that point is also checked to see if it occurs on a horizontal grid line. If that does not occur the whole procedure is repeated for the next interval beginning with the calculation of the grid intersect on the vertical grid line which *is now the lower bound of the new interval*.

Calculating the Nodes of the Quantization

The basic structure of the algorithm which is used to calculate the nodes of the quantization is similar to that for the quantization procedure presented in Chapter II. In Chapter II, however, the input to the procedure was a line drawing superimposed on a grid; whereas, in this case, the input is a sequence of grid intersects.

As before, the first quantized node is that one which is closest to the first grid intersect. The procedure for selecting next nodes is then repeated until all of the nodes of the quantization have been calculated. Before a possible next node can be found and validated, an elimination of any redundant grid intersects must be performed. Redundant intersects occur when two or more successive grid intersects are within half the grid size of the same grid node. The necessity of this elimination process is

illustrated in the example shown in Figure 3-3. In this example, the node E is the current node and the grid intersect A is the one which was used in selecting it. If the (1,2)-code is being used the next node selected should be the node F. The validation of this node, which checks all intersects between the one used in selecting the current node and the one on the ring to see if they lie within the area of precision, would fail because of the redundant grid intersect B. Due to the nature of the definition of the area of precision and to only possible location of redundant grid intersects, these intersects will always be outside the area of precision and, thus, need to be eliminated. Redundant intersects are eliminated by redefining the grid intersect which is associated with

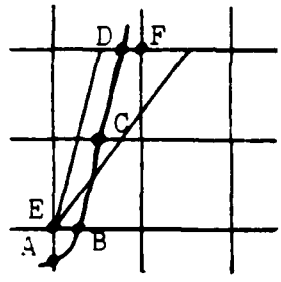


Figure 3-3 Example of Redundant Grid Intersects

the current node to be the last redundant intersect. For the example in Figure 3-3, the grid intersect B is considered to be the one used to calculate the node E, so that the validation procedure will only test the grid intersect C in the validation of node F.

Once the elimination of redundant nodes has been accomplished, the selection of the next node can be made. Since the path of the line drawing cannot be traced, the sequence of grid intersects must be searched to find the grid intersect on a particular ring. The search is made by testing

each grid intersect after the one associated with the current node until the first one which lies on the ring is found. The test is performed by calculating the difference between the coordinates of the grid intersect and the corresponding coordinates of the current node. If either, or both, of these differences is equal to the product of the ring number and the grid size, then the grid intersect lies on the ring.

After the search is completed, the validation of the possible next node is performed. Since the grid intersect cannot be visually tested to see if they are located within the area of precision, this validation process must be implemented in another manner. This is done by utilizing the fact that the area of precision represents an angular range within which the grid intersects must lie for a successful validation. The lower bound of this range is defined as the angle between the line segment from the current node to the point on the ring one-half the grid size clockwise from the possible next node and the positive x-axis. The upper bound of the range can similarly be calculated. This validation angle range is illustrated in Figure 3-4. The validation process now consists of calculating the angles associated with each appropriate grid intersect and comparing these with the range just calculated.

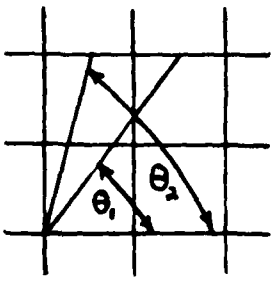


Figure 3-4 Angle Range for Node Validation

The rest of the algorithm for calculating a particular node of the quantization is the same as given in Chapter II. When the end of the sequence of grid intersects is encountered, the search and validation process may not be appropriate for some of the outer rings. In this case, the process begins with the outermost ring on which a grid intersect lies.

Calculation of the Performance

In this study, two measures of performance were calculated: the area error between the line drawing and the quantized approximation and the encoding rate. Each of the measures was scaled by the total path length of the line drawing. There are three computations which must be performed in the calculation of the performance of the code. These are the calculation of the path length of the line drawing, the integration of the error between the line drawing and the quantized approximation to it, and the number of bits which would be used to encode the quantization.

The path length of the line drawing is calculated by the integral:

$$\int_D \left[1 + (f^1(x))^2\right]^{\frac{1}{2}} dx \qquad (3.1)$$

where f(x) is the function defining the line drawing and D is its domain.

The calculation of the area error is performed by integrating the difference between the line drawing and the quantized approximation of the line drawing. The quantized approximation of the line drawing is composed of a set of line segments connecting the nodes of the quantization. If there are N nodes in the quantization, the integration is broken down into N-1 integrals. Since the integrand can change sign, the absolute value of

the integrand is integrated. The calculation of the area error can be expressed as follows:

$$\text{area error} = \sum_{i=2}^{N} \int_{X_{i-1}}^{x_i} \left| f(x) - \ell_i(x) \right| \, dx \qquad (3.2)$$

$$\text{where } \ell_i(x) = \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \left(x - x_{i-1} \right) + y_{i-1}$$

is the equation of the line between nodes i and i-1.

Once the area error has been calculated, it is divided by the path length to give the error performance measure of the code.

The rate performance measure is computed by calculating the total number of bits that would be necessary to encode the quantization and dividing this result by the path length. The total number of bits is equal to the number of line segments in the approximation (one less than the number of nodes in the quantization) multiplied by the number of bits required to encode each segment. The number of bits per segment is calculated by first calculating the number of possible next nodes. For any ring N, the number of possible next nodes on that ring is 8N. The total number of possible next nodes is the sum of the ring numbers of the code multiplied by eight. To determine the encoding rate per segment the base 2 logrithm of the total number of bits is computed and any fraction of the result is rounded to the next larger integer. This result is multiplied by the number of segments and divided by the path length to give the encoding rate per length of the line drawing.

Summary

In this chapter the methods used in this study to simulate and analyze the generalized chain codes were presented. The analysis was divided into three sections: the calculation of the grid intersects, the calculation of the nodes of the quantization, and the calculation of the performance. In the next chapter, the software implementation of these methods will be presented.

IV. Software

In this chapter the structure and design of the software used to implement the analysis presented in the last chapter is discussed. The program is designed to simulate the line drawing with either a sine wave or a periodic circular wave (defined later in the chapter). The program is also designed to implement the quantization procedure of any form of the generalized chain code. The description of the program will consist of a discussion of the overall operation of the program, a discussion of the subprograms used to simulate the line drawing, and a brief discussion of the subprograms used to perform each section of the analysis. The flowcharts presented in Appendix B illustrate the software implementation discussed in this chapter. The validation of the software was performed by running several test cases and comparing the results to the results derived by performing the same analysis by hand.

Overall Program Operation

The overall program operation consists of reading the input parameters of the particular test, calculating the grid intersects, calculating the quantization nodes, calculating the performance measures, and outputting the results.

The inputs to the program are the parameters which define the function, the parameters which define the code, the grid size, and the number of periods of the function over which the analysis is performed. The function parameters are a function number (1-sine wave, 2-periodic

circular wave), the period, the initial phase, and the amplitude. For the periodic circular wave, the amplitude is not read as an input but is calculated as one-fourth of the period. In order to prevent the instantaneous phase at any two vertical grid lines from being the same, the value of $\pi \times 10^{-3}$ was added to the period. The code parameters are the number of levels of the code and the ring numbers of the code.

After the inputs have been read, the analysis is performed as discussed in the previous chapter. Since the coordinates of the grid intersects and the quantization nodes are stored in arrays, the computation of all of the grid intersects before the calculation of the quantization nodes could require an excessive amount of core memory. In order to prevent such a requirement, the analysis is performed as follows. A certain number of intersects are calculated. These values are stored in corresponding elements of two arrays. The quantization nodes are determined for the elements of the array up to the point where only sufficient array elements are left to calculate one more node. At this point the remaining elements are moved to the beginning of the array. Starting with the last grid intersect calculated, the array is once again filled. This procedure continues until the end of the domain of the function is reached.

The performance measures are calculated at each point which is a multiple of ten times the period of the function. At each of these points the accumulated error per unit length and the encoding rate per unit line is calculated, thus, providing the asymptotic characteristics of the measures.

Simulation of the Line Drawing

There are three functions which are required for use in simulating the line drawing: the function defining the line drawing, its derivative, and its inverse. These functions are implemented in the function subprograms F and DF, and the subroutine FINV. In each of these subprograms, both the sine wave and the periodic circular wave are implemented. The functions are distinguished in the code by a "computed GO TO" statement based on the function number which was an input to the program. The implementation of each of the functions in the subprograms will now be given.

The Sine Wave. The sine wave and its derivative are easily implemented with a simple FORTRAN expression using the intrinsic FORTRAN functions, SIN and COS. The implementation of the inverse function is more difficult since the function is not uniquely invertible. This difficulty can be surmounted as a result of the fact that the inverse is only required for the calculation of the grid intersects. The inverse is implemented by using the knowledge of the location of the previous vertical grid line and the knowledge of the possible occurrence of a peak on the function within an interval.

The first problem is to determine how many full periods have occurred preceding the current point. This is done by dividing the coordinate of the preceding vertical grid line by the period and truncating the result. The intrinsic FORTRAN function for implementing the inverse of the same function is then evaluated. Another problem arises here in that this function only returns values from $^{-\pi}/_{2}$ to $^{+\pi}/_{2}$. Therefore, this value must be adjusted based on the knowledge of the quadrant in which the point for which the inverse is calculated occurs.

The quadrant is determined by dividing the modulus of the division operation previously performed by one-fourth of the period and truncating the result. This value is incremented by one if a peak or x-axis crossing occurs. If the resultant value is zero then the value is set to one. If the value is greater than four, then the number of periods is incremented by one and the quadrant number is set equal to four less than the value indicated. The intrinsic function, ASIN, is then called. If the first quadrant has been indicated, the value returned is the correct value. If the second or third quadrant has been indicated, the value return is increased by π. For the fourth quadrant, the value is increased by 2π. This adjusted value is then added to the product of the number of periods times the period to give the actual value of the inverse.

The Periodic Circular Function. The periodic circular function is defined by the equation:

$$f(x) = (-1)^N \left[A^2 - (X + \Theta - (2N + 1)\, A)^2 \right]^{\frac{1}{2}} \qquad (4.1)$$

$$2NA \leq X < (2N + 2)A$$

$$N = 0, 1, 2, \ldots$$

where A is the amplitude and Θ is the initial phase. The function is illustrated in Figure 4-1. The period of this function is 4A. The shape of this function can be visualized by imagining a series of circles of diameter 2A centered on the x-axis and separated by a distance of 2A. The lower half of these circles are then shifted a distance of 2A to the right; thus, connecting the waveform. Implementing

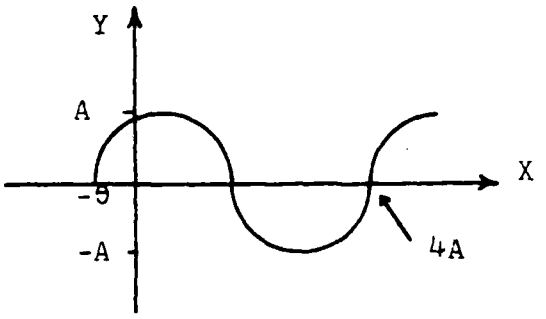


Figure 4-1 The Periodic Circular Function

this function involves calculating the value of N and then evaluating the expression. In calculating the value of N, the sum of the argument and initial phase divided by half the period and the result truncated.

The derivative of this function can be expressed by the equation

$$f^1(X) = \frac{(-1)^M \left| X + \Theta - (2N + 1)\, A \right|}{\left[A^2 - (X + \Theta - (2N + 1)\, A)^2 \right]^{\frac{1}{2}}} \qquad (4.2)$$

$$2NA \leq X < (2N + 2)\, A$$

$$N = 0, 1, 2, \ldots$$

where A and Θ are the same as before and M is defined such that the function will be positive in the first and fourth quadrants and negative in the second and third. To implement this function, the values of N and M must be calculated and then the expression evaluated. The calculation of M is performed by dividing the sum of the argument, the initial phase, and one-fourth of the period by half the period and the result truncated.

The task of implementing the inverse function is plagued by the same problem as implementing the inverse of the sine wave: the periodic circular function is not uniquely invertible. The equation for the inverse function is:

$$X = (-1)^M \left[A^2 - Y^2 \right]^{\frac{1}{2}} - \theta + (2N + 1)\, A \qquad (4.3)$$

for $y = f(x)$ and where A, θ, and N are the same as before and M is defined such that it will be an odd integer if the point where the inverse is calculated is located in the first or third quadrant and will be an even integer if the point is located in the second or fourth quadrant. The knowledge of the location of the previous vertical grid line and the occurrence of a peak or zero crossing between the vertical grid line and the current point is used in calculating M and N. The value of M is calculated as the total number of quarter periods up to and including the one in which the current point is located. This is done by dividing the coordinate of the vertical grid line by one-fourth of the period and truncating. To this value is added the number of occurrences of a peak or zero crossing to give the value of M. The value of N is the total number of half periods up to and including the one in which the current point is located. This value is calculated by dividing the sum of coordinates of the previous vertical grid line and the initial phase by half the period. To this result is added the number of zero crossings between the grid line and the current point. The expression for the inverse of the periodic circular wave can then be evaluated.

The Calculation of the Grid Intersects

The calculation of the grid intersects is performed by the subroutines FIP and CIBD. The subroutine FIP is called when grid intersect calculations are to be performed. The inputs to these subroutines are the location of the first grid intersect, the subscript of the first available location in the arrays in which the coordinates of the grid intersects are stored, and the number which limits the number of grid intersects which may be calculated at one time. The latter number is computed in the main program as one less than the dimension of the arrays containing the grid intersect minus twice the amplitude of the function. For the first time that FIP is called the other two inputs are zero and one, respectively. Thereafter, the location of the next grid intersect is determined by where the preceding set of grid intersect calculations was interrupted and the array location parameter is determined by the results of the quantization procedure for the previous set of grid intersects.

FIP. The first task of subroutine FIP is to initialize control and accounting variables. The first of these is that of the variable JFIN. This variable is a flag used in controlling subsequent calls to the subroutine. The initial value of this flag is 1. If the grid intersects arrays are filled before the end of the domain of the function is encountered, the flag is set equal to 0. This will result in another call to this subroutine after the current set of grid intersects have been processed.

The next initialization to be made is that of the variable NIP. This variable is a pointer for the arrays XINT and YINT which contain the coordinates of the grid intersects. During the calculation of the

grid intersects, this variable will always point to the location in the arrays where the next grid intersect will be stored. When the return from this subroutine is executed, this variable will point to the last grid intersect in the arrays indicating the total number of grid intersects in the arrays. The initial value of this variable is passed to the subroutine via the parameter NIPST. When the first call to FIP occurs NIP will be 1. For subsequent calls however the value will be determined by the results of the calculation of the quantized nodes.

The final initialization is that of the variable T1. This variable contains the value of the lower bound of the first interval. The initial value of this parameter is passed to the subroutine via the variable TSTART. Its value will be zero for the first call to this subroutine, but for future calls will depend on the last grid intersect calculated on the previous call. The grid intersect corresponding to T1 will always be the first calculated when the subroutine is called.

The procedure for calculating the grid intersect is implemented as follows. The value of the function and its derivative at T1 are first calculated and stored in FT1 and DFT1, respectively. Due to the repetitive nature of the algorithm, the calculation of the intersects is made in a loop. One pass through the loop results in the calculation and storage of the intersects on the first vertical grid line and those on horizontal grid lines occurring before the next vertical grid line. The loop begins by storing the intersect on the first vertical grid line (i.e., T1 and FT1 into XINT(NIP) and YINT(NIP), respectively). A check is then made to determine if XEND has been reached. If T1 equals XEND then a return is executed. At this point JFIN is still 1 so the main program will "know" that the last grid intersect has been calculated.

If XEND does not equal T1, the pointer NIP is incremented. At this time the variable, NQCF is set to zero. This variable is used by the subroutine FINV to calculate the inverse of the function F as a counter of the possible peaks and/or zero crossing in an interval. The value of the abscissa at the next vertical grid line is then calculated by adding the grid size to the value of T1. This value is stored in T2. A check is made to determine if the value of T2 exceeds that of XEND. If it does, an exit from the loop is made and the intersects of that final interval are calculated. This will be discussed in more detail later. If T2 is less than or equal to XEND the value of the function and its derivative at T2 are calculated and stored in FT2 and DFT2. The grid intersects on horizontal grid lines are now to be calculated. Due to the periodic nature of the function, this is not a simple task. The method used is to divide the interval into subintervals in which the slope is strictly increasing or strictly decreasing. This is accomplished by first determining if it is necessary to divide the interval, that is, to determine if the slope changes sign. This test is made by calculating the sign of DFT1 and DFT2 and adding them together. This sum is stored in the variable JTEST. A value other than zero indicates that no division of the interval is required. A call to the subroutine CIBD is then made which calculates the intersect occurring within the total interval. A value of zero will require a division of the interval. Due to the nature of the subroutine CIBD, only the ordinate values and the derivatives at the endpoints of the subinterval must be supplied. Since these values are already known at T1 and T2, it is only at the point of division where they must be calculated. The derivative at this point is zero by definition. The ordinate at this point (YD) will be either

+A or -A. The amplitude is known so that only the sign must be determined. This is done by calculating the sign of DFT1 since if the slope is increasing (decreasing) in the first subinterval the ordinate will be positive (negative). Though it is possible for the sign of DFT1 to be zero, this is not a problem since in that case there would be no need to divide the interval because the grid size is at most one-quarter of the period. Once the dividing point is determined, the subroutine CIBD is called for the first subinterval. This subroutine calculates and stores the intersects on horizontal grid lines within the interval and not the possible one at YD. Therefore, a check must be made to determine if the function intersects a horizontal grid line at YD. This is done by first dividing YD by the grid size and storing this value in PIT. PIT is then truncated and compared with its original value. If these two values are not the same then there is no intersect. If they are the same, the inverse of the function (FV) is calculated by a subroutine call to FINV. FV and YD are then stored in XINT(NIP) and YINT(NIP), respectively, and NIP is incremented. At this point the variable NQCF is incremented and then CIBD is called again to calculate the intersects in the second subinterval. All of the intersects have been calculated for this pass through the loop. The variables T1, FT1, and DFT1 are set equal to T2, FT2, and DFT2. Before returning to the starting point of the loop, a check is made to see if NIP exceeds NLIMIT. If it does not then the loop begins again. If it does then JFIN is set to zero, TSTART is set equal to T1, NIP is decremented, and a RETURN is executed.

Early in the loop a check was made to see if T2 exceeded XEND. The previous discussion describes the results if T2 was less than XEND.

If T2 does exceed XEND, an exit from the loop is made to a point where much of the logic of the loop is repeated except for the following differences. First, T2 is reset equal to XEND and FT2 and DFT2 are recalculated for this point. The rest of the logic is the same until the end. After CIBD is called either for the whole interval or for the second subinterval, a check must be made to determine if there is an intersect with a horizontal grid line at XEND. This is done in the same manner as the test at YD. After this test is made a RETURN is executed. The value of JFIN will be one indicating that the last grid intersect has been calculated.

CIBD. As previously stated, the subroutine CIBD is used to calculate the intersect points on the horizontal grid line within some interval. The ordinate values and the derivatives at the endpoints of this interval are passed to the subroutine through the parameters of the CALL statement and are given the variable names Y1, DY1, Y2, and DY2. The grid intersects are calculated in a DO-loop in which the counter (JC) points to the horizontal grid line where a grid intersect is located. That is, if JC equals 1, the counter points to the first horizontal grid line above the abscissa (note: if the grid size (DEL) equals 1, JC equals the ordinate value of the horizontal grid line).

Before this loop can be executed the parameters of the loop must be determined. These parameters are the lower (JFY) and upper (LY) limits of the counter, and the increment (MD). The first parameter that is calculated is the increment. This is done by calculating the sign of the sum of DY1 and DY2. Since DY1 and DY2 will either both have the same sign or one of them will be zero, MD will be +1 if the slope of the function in the interval is increasing and -1 if the slope is

decreasing. The calculation of the parameters, JFY and LY, are more difficult to make because the calculation depends on the sign of MD, Y1, and Y2. The logic that performs this task is composed of a dozen logical IF statements for which a detailed explanation would be tedious and unnecessary. The result is that if the slope is positive (negative), then JFY corresponds to the first horizontal grid line above (below) Y1 and LY corresponds to the first horizontal grid line below (above) Y2.

After the parameters of the loop are calculated, the DO loop is executed. It is important to note that, as a result of the manner in which the parameters of the loop were calculated, the counter points to each horizontal grid line in the order that they would be encountered if the path of the function were traced from Y1 to Y2. The first operation to be performed inside the loop is to calculate the y-component of the intersect on the appropriate horizontal grid line. This is done by multiplying JC by the grid size. The x-coordinate of the intersect is then found by a call to the subroutine FINV. At this point, a test is made to see if JC equals zero and if so, the variable NQCF is incremented. Finally, the pointer NIP is incremented. Upon completion of this loop, all of the intersects in the interval have been computed and a RETURN to FIP is executed.

Calculation of the Nodes of the Quantization

The calculation of the quantized nodes is performed by a set of five subroutines: GERD, FNSRI, FNGN, FAR, and ANGLE. These routines are independent of the waveform for which the grid intersects were calculated. The subroutine GERD implements the quantization scheme, while the other four perform related tasks. The subroutine FNSRI

locates the first grid intersect on a particular code ring. If the last grid intersect lies within the code ring, however, a zero is returned. The subroutine FNGN calculates the coordinates of the grid node closest to a grid intersect. The subroutine FAR calculates a range of angles which is used in the validation of a possible next node of the quantization. Finally, the subroutine ANGLE calculates the angle between the positive x-axis and the line segment defined by the center of the code rings and some point on the grid. The range of this angle is between $\emptyset$ and 2π radians. The purpose of these subroutines will be made more clear in the description of the subroutine GERD.

GERD. This subroutine takes the grid intersects and calculates the encoded nodes. Since the procedure for calculating a quantized node is dependent on the previous node, the first node is chosen as the grid node closest to the first grid intersect. The subroutine FNGN is called to calculate the coordinate of this node. The procedure for calculating the next node is then initiation. This procedure is repeated for each subsequent node of the quantization and so is implemented in a loop.

The first step in this loop is the elimination of redundant grid intersects as described in the last chapter. The next step in the subroutine is to set up the accounting logic which keeps track of the code level and the ring number at that level. The ring numbers are stored in the array NRL, thus the subscript of the array corresponds to the code level. To initialize the code level pointer (LCP) to the highest level, it is set equal to the total number of levels. A ring pointer (NRS) is then set equal to the ring number in the NRL array

location LCP. This is the point to which the subroutine returns if the validation of a possible next node fails.

The search for the grid intersect on the ring NRS is now initiated. This is done by a call to the subroutine FNSRI. There are two possible results of this subroutine call. The first is that all of the grid intersects which have thus far been calculated may lie inside the ring. In this case, the flag JFIN is checked to determine if more intersects are to be calculated. If more intersects are to be calculated, no further quantization can be done and an exit from GERD is executed. If no further intersects are to be calculated, the code level pointer is checked to determine if the current code level is the lowest of the code (i.e., LCP = 1). If LCP equals unity, the grid node closest to the last intersect is calculated. This node is included in the quantized sequence if it lies on the inner ring of the code. (Note: This is possible even if the intersect does not lie on the ring.) If LCP is greater than unity, it is decremented and the subroutine returns to the point where the ring pointer is set so that the next lower level of the code can be processed.

The other possible result of the call to FNSRI is that an intersect is found on the ring. The grid node closest to this grid intersect is then calculated by FNGN. The code level pointer is then checked to determine if the ring is the innermost one of the code in which case the node is stored as a quantized node. If the code level pointer indicates that the ring is not the innermost one, the procedure for validating the node is initiated.

This procedure begins with a call to subroutine FAR to calculate the range of angles used in the validation. The angle corresponding to

the grid intersect immediately preceding the one on the ring is first calculated. This angle is compared with the range of valid angles. If it lies within the range, the next inner grid intersect is processed in the same manner. If the angle does not lie within the range, the code level is decremented and the subroutine returns to the point where the ring number is set so that the next inner ring can be processed. If all of the intermediate grid intersects lie within the range, the node on the ring is stored as a quantized node. A check is then made to determine if the grid intersect associated with the new node is the last one thus far calculated in which case an exit from GERD is executed. Otherwise, the subroutine returns to the beginning of the loop to begin the procedure to calculate the next node of the quantization. The procedure is repeated until the last grid intersect has been processed.

FNSRI. The algorithm used in this subroutine to calculate the first grid intersect on a ring is a comparison of the difference between the coordinates of a grid intersect and the coordinates of the center of the ring. If the difference between either or both the x-coordinates or y-coordinates is equal to the distance to the ring, then the intersect is the one on the ring. The first grid intersect to be thus tested is the one whose subscript is equal to the sum of the subscript of the last grid intersect to be encoded and the ring number. This is the first grid intersect which can possibly be on the ring. If this intersect fails the test, the next grid intersect is tested. This procedure is repeated until either an intersect on the ring is found or the last grid intersect fails the test. In the first case, the subscript of that grid intersect is returned to GERD. In the second case, zero

is returned instead of a valid subscript indicating that an intersect was not found on the ring.

FNGN. The subroutine FNGN calculates each coordinate of the grid node closest to a grid intersect separately. The method used is the same for both coordinates. First the coordinate of the grid intersect is scaled down by the grid size. The modulus of the result is then extracted by subtracting the integer portion from the total. If the fraction is less than or equal to one-half, the coordinate of the grid node is the integer portion of the scaled coordinate of the grid intersect multiplied by the grid size. If the fraction is greater than one-half, the coordinate of the grid node is the integer portion of the coordinate of the grid intersect increased in magnitude by one and then multiplied by the grid size.

FAR. The subroutine FAR first calculates the point one-half of the grid size clockwise of the grid node and the point one-half of the grid size counter-clockwise of the grid node. The subroutine ANGLE is then called for each of these points to calculate the angles associated with them. The calculation of these points is dependent on the location of the grid node on the ring.

The grid node is first tested to determine if it is one of the four corner nodes of the ring. This is performed by evaluating the difference between each coordinate of the grid node and the corresponding coordinate of the center of the ring (hereafter, referred to as the x-difference and the y-difference). If the magnitude of both of these differences are equal to the ring number multiplied by the grid size, then the grid node must be a corner node. The corner nodes which are diagonally across from each other can be processed using the same equation. To thus

distinguish these nodes, the product of the signs of the x-difference and the y-difference is evaluated. If this product is positive, then the node is either in the first or third quadrant of the ring. The point corresponding to the lower bound for this case is the point whose x-coordinate is the same as the x-coordinate of the grid node and whose y-coordinate is the y-coordinate of the grid node less the product of the sign of the y-difference and half the grid size. The point corresponding to the upper bound is the point whose x-coordinate is the x-coordinate of the grid node less the product of the sign of the x-difference and half the grid size and whose y-coordinate is the same as the y-coordinate of the grid node. The bounds are calculated in a similar manner in the second and fourth quadrants.

For grid nodes other than corner nodes, the calculations are dependent on whether the node is located on a vertical side of the ring or on a horizontal side of the ring. This determination is made by subtracting the magnitude of the y-difference from the product of the ring number and the grid size. If the result is zero, then the node must lie on a horizontal side of the ring. For this case, the y-coordinate for both the upper and lower bounds is the same as the y-coordinate of the grid node. The x-coordinate of the upper (lower) bound is the product of the sign of the y-difference and half the grid size subtracted from (added to) the x-coordinate of the grid node. If the node is located on a vertical side of the ring, the x-coordinate of both the upper and lower bounds is the same as the x-coordinate of the grid node. The y-coordinate of the upper (lower) bound is the product of the sign of the x-difference and half the grid size added to (subtracted from) the y-coordinate of the grid node.

ANGLE. The subroutine ANGLE calculates the angle associated with a point on the grid by calculating the inverse tangent of the slope of the line segment defined by the center of the code rings and the point. Since the slope can be infinite, the x-coordinate of the point is first compared to the x-coordinate of the center of the rings. If the slope is infinite, the y-coordinate of the point is compared to the y-coordinate of the center of the code rings. If the former is greater, the angle is set equal to $\pi/2$ and if the latter is greater, the angle is set equal to $3\pi/2$.

If the slope is not infinite, the FORTRAN intrinsic function ATAN can be used to calculate the inverse tangent. The function calculates angles in the range between $-\pi/2$ and $\pi/2$. This creates a problem if the point lies in the second or third quadrant. To account for this problem, the quadrant in which the point lies is first determined. This is accomplished by comparing the coordinates of the point with the coordinates of the center of the code rings. If the point is in the first quadrant, the angle is equal to the inverse tangent of the slope as calculated by ATAN. In the second quadrant, the angle calculated by ATAN (negative) is added to π. In the third quadrant this angle is added to π to determine the correct angle. Finally, in the fourth quadrant, the angle calculated by ATAN (negative) is added to 2π.

Subroutines Used in Calculating the Performance

There are eight subroutines which aid in the calculation of the performance of the code including: CAE, CAESL, FDE, DEDE, ZERO, CLI, FDLI, and SPGQI. The purposes of these routines are to calculate the area error in the approximation to the waveform over some arbitrary

interval and to calculate the path length of the waveform over some arbitrary interval.

In calculating the area error, the interval is first divided into subintervals corresponding to the individual line segments of the approximation since the integrand of equation (3.2) may not be the same over the entire interval. These subintervals are further divided into ranges over which the integrand is strictly positive or strictly negative to prevent unwanted cancellation. The integration is then performed for each of these small ranges and the absolute value of the result added to an accumulator. The calculation of path length is performed by dividing the interval into subintervals small enough to ensure the accuracy of the numerical integration procedure. Since the integrand of this integral is always positive, the integration can then be performed and the results for each subinterval added together.

CAE. The purpose of the subroutine CAE is to divide the interval into subintervals corresponding to the line segments of the approximation and to initiate the next step in the area error calculation. This is the subroutine called when an area error calculation is to be made. The first step in accomplishing this purpose is to calculate the subscript (JFN) of the first quantized node occurring after the lower limit. The subscript (JLN) of the first node preceding the upper limit is then calculated. The subscripts, JFN and JLN, are then compared. If JLN is less than JFN, then the entire interval corresponds to one line segment of the approximation. The subroutine CAESL is then called to process this interval after which a RETURN is executed. If JLN is not less than JFN, CAESL is called to process the interval between the lower limit and the quantized node corresponding to JFN. The interval between the node

corresponding to JLN and the upper limit is then similarly processed. At this point JFN and JLN are again compared. If they are equal, then the entire interval has been processed. Otherwise, there exist one or more intervals between the two just processed which correspond to complete line segments of the approximation. These line segments are then processed in a DO loop in which vertical line segments are ignored.

CAESL. The purpose of the subroutine CAESL is to divide the interval relayed from CAE into subintervals in which the integrand is strictly positive or strictly negative. The first step in accomplishing this purpose is to calculate the signs of the integrand and its derivative at both the upper and lower limit of the interval. By a series of comparisons of these signs, the number of times the integrand is zero within the interval is determined. Using the subroutine ZERO, the location of these zeros are found. These zeros define the subdivisions of the interval. The subroutine SPGQI is then called to perform the numerical integration over the subdivisions. The sum of the absolute value of the result of each integration is then computed and returned to CAE.

FDE and DFDE. The function subprogram FDE computes the value of the integrand at some time. The function subprogram DFDE computes the derivative of the integrand at some time.

ZERO. The subroutine ZERO locates the zero of a given function within a given interval. To perform this calculation, the modified regula falsi algorithm is implemented. A complete discussion of the algorithm can be found in Conte and de Boor [2]. This algorithm was selected because it is relatively fast and stable (in the sense that the search always stays within the interval). The convergence criterion implemented is two-fold. If either the point where the

function is less than 10^{-5} or the width of the search interval is less than 10^{-5} the algorithm is considered to have converged.

CLI. The subroutine CLI is called to perform the calculation of the path length of the waveforms over a particular interval. If the waveform is the periodic circular function, the line integral can be calculated directly as the product of the width of the interval and $\pi/2$. If the waveform is the sine wave, numerical integration must be used. The integrand is always positive and so that interval is only divided into intervals small enough to ensure the accuracy of the numerical integration.

FDLI. The function subprogram FDLI calculates the value of the integrand of the integral given in equation (3.1) at a given point.

SPGQI. The subroutine SPGQI is called to perform the numerical integration of a particular function over a given interval. The algorithm implemented to accomplish this is the six-point Ganssian quadrature integration algorithm. A complete discussion of this algorithm can also be found in Conte and de Boor [2]. The nodes and weights for the interval [-1, 1] are supplied. An appropriate transformation of the integral of the given range is made so that the integration can be performed over the interval [-1, 1].

V. Results and Recommendations

In this chapter, the results of this study are presented and discussed, and recommendations for future study are made.

Results

The results are illustrated in Figure A-1 through A-24 of Appendix A. In each of these figures, the upper plot is that of the accumulated encoding rate per unit length of the line drawing versus the domain of the function defining the line drawing. The lower plot is that of the accumulated area error per unit length of the line drawing versus the domain of the function defining the line drawing.

The results can be divided into three types of tests:

1. Comparing the performance of a code for a function at different values of the initial phase.
2. Comparing the performance of different codes for the same function.
3. Comparing the performance of a code for both the sine wave and the periodic circular wave, each having the same amplitude, period and initial phase.

In the following paragraphs, the figures associated with each test will be identified and discussed.

Same Code, Same Function, Different Phase. The purpose of these tests was to study the effect of the initial phase of a function on the quantization and encoding performance. In each test, five values of initial phase were used: Ø.Ø, Ø.2, Ø.4, Ø.6, and Ø.8. Figures A-1

through A-4 illustrate the results for a sine wave of period 20 and amplitude 5 encoded by the (1)-, (1,2)-, (1,3)-, and (1,2,3,)-code, respectively. Figures A-12 through A-15 illustrate similar results for the periodic circular wave with period 20 (and, therefore, amplitude 5).

In each test, the performance in both rate and error converged, in the limit, to the same values for each of the different values of the initial phase. This observation indicates that the asymptotic value of the performance is independent of the initial phase (at least for the sine wave and the periodic circular wave).

It is interesting to note that, in each plot, the performance curves for all values of initial phase intersect at three different times. These points correspond to the vertical grid lines where the phase is very nearly the same as the initial phase. To substantiate this claim, recall that a value of $\pi \times 10^{-3}$ was added to the period. Since the grid size is unity, the number of periods which would elapse before the phase at a vertical grid line would be almost the same as the initial phase is equal to the reciprocal of $\pi \times 10^{-3}$, or approximately 318 periods. Consider, for example, the case for Figure A-1. In this case, the period is $20 + \pi \times 10^{-3}$ and the first intersection of the error performance curves occurs at approximately $x = 6300$. The product of 318 times the period is 6350 which is very close to the value estimates from the plot.

Different Codes, Same Function. The purpose of these tests was to compare the performance of different codes used to quantize and encode the same function. The codes used in these comparisons were the (1)-, (1,2)-, (1,3)-, and the (1,2,3,)-code. Figure A-5 through A-11 illustrate the results for sine waves of amplitude 5, initial phase, 0, and periods 5, 10, 15, 20, 30, 40, and 60, respectively. Figures A-16 through A-20

illustrate similar results for periodic circular waves with periods 10, 20, 30, 40, and 50, respectively.

In each of these tests, the (1)-code consistently exhibited the worst error performance of any of the codes implemented. This indicates that the limited angular resolution of this code significantly reduces its ability to follow the curvature of the line drawing closely. This effect was not as consistent (in the sine wave tests) with the (1,2)-code because the difference in angular resolution between this code and the higher order codes is not as great as the difference for the (1)-code.

The comparative rate performance for the (1)-code was not constant with respect to the rate performance of the other codes. For the sine waves of the smallest and larger periods, the rate performance for this code was worse in comparison to the other codes. However, for the middle periods, the performance of the (1)-code improves with respect to the other codes. This indicates that the relative performance of this code is dependent on the amount of relatively straight portions of the line drawing. For line drawings which have longer segments of relatively straight lines (such as, sine waves with very long periods or high amplitudes), the higher order codes are able to utilize their outer rings. As a result, fewer code words are required and the encoding rate is improved. Conversely, for line drawings with more curvature, the higher order codes cannot utilize their outer rings; thus, more code words are required and the rate performance is degraded.

Other observation can be made about the performance of the (1)-code. The amount of fluctuation in the rate performance curves for this code was consistently smaller than for the other codes. This was not surprising in view of the simplicity of the code. A more significant

observation about the rate performance of the (1)-code is that the asymptotic value of the rate for the periodic circular wave was virtually the same for each different period. None of the other codes exhibited this consistency in rate performance for the constant curvature (except at the zero crossings) of the periodic circular wave.

Observations and conclusions were harder to draw from the higher order codes. In some cases, the results were unexpected. For example, in Figure A-7, the performance in both rate and error was better for the (1,2)-code than for either the (1,3)- or (1,2,3)-code. One would expect that better error performance would be the result of finer angular quantization which would result in poorer rate performance. It is apparent that the higher two codes were not able to make use of their outer ring consistently. As a result, the error performance could at best be only as good as for the (1,2)-code, and actually was worse. Since the higher codes require more bits per code word, the rate performance of the two higher order codes likewise suffered.

A couple of final observations can be made about the trends in the results of this set of tests. For example, for the three largest periods for the sine wave, the relative rate performance of the four codes was the same. That is, if the codes are ranked by encoding rate, the order is the same for each case cited. Another trend occurs in the error performance for the ciruclar wave. In fact, for every period tested, a ranking by error performance was the same order in each case.

Same Code, Different Function. The purpose of these tests was to compare the effect of constant curvature on each code. The results of these tests are illustrated in Figures A-21 through A-24. In each case, the code performed better for the sine wave, the result more pronounced

for the (1)- and (1,3)-codes and the difference was negligible for the error performance of the (1,2,3)-code.

Recommendation for Future Study

This study was by no means complete. There are many interesting possibilities for future investigation. The following are my recommendations:

1. Evaluate the performance of the codes for sine waves of different amplitudes.
2. Evaluate the performance of the codes for either function for different periods.
3. Plot the asymptotic values of the performance measures versus the ratio of period to amplitude.
4. Redesign the software to implement the quantizing in terms of the first grid intersect within half the grid size of a ring instead of the first intersect on the ring and compare the performance of each.
5. Using performance data from the circular wave, plot the performance versus radius of curvature.

Bibliography

1. Castor, K. G. and Neuhoff, D. L. "A Rate and Distortion Analysis for Grid Intersect Quantization of Line Drawing Images," Proceedings of the 18th Annual Allerton Conference on Communication, Control, and Computing, University of Illinois at Urbana-Champaign, October 1980.

2. Conte, S. D. and de Boor, C. Elementary Numerical Analysis (Third Edition), New York: McGraw-Hill Book Company, 1980.

3. Freeman, Herbert. The Generalized Chain Code for Map Data Encoding and Processing. Technical Report CRL-59. Air Force Office of Scientific Research. June 1978.

4. Freeman, Herbert and Saghri, A. "Generalized Chain Codes for Planar Curves," Proceedings of the 4th International Joint Conference on Pattern Recognition. Kyoto, Japan. November, 1978.

Appendix A

Performance Plots

In each figure, the upper plot is that of the accumulated encoding rate per unit length versus the domain of the function. The lower plot is that of the accumulated area error per unit length versus the domain of the function. Each of the performance curves in these plots is distinguished by a letter and a number. The letter identifies the function implemented for the line drawing, as well as, the coding schemes used to encode it. The letters A, B, C, and D represent the (1)-, (1,2)-, (1,3)-, and (1,2,3)-code, respectively, for the sine wave. The letters E, F, G, and H represent the same codes for the circular wave. The number gives the initial phase in tenths of a radian. The table on the following page identifies each figure in terms of the function and its parameter, and a brief description of the comparison illustrated.

Table of Figures

Figure Number	Function	Amplitude	Period	Description
A-1	Sine	5	20	Five different values of initial phase, (1)-code
A-2	Sine	5	20	Five different values of initial phase, (1,2)-code
A-3	Sine	5	20	Five different values of initial phase, (1,3)-code
A-4	Sine	5	20	Five different values of initial phase, (1,2,3)-code
A-5	Sine	5	5	Four different codes, zero initial phase
A-6	Sine	5	10	Four different codes, zero initial phase
A-7	Sine	5	15	Four different codes, zero initial phase
A-8	Sine	5	20	Four different codes, zero initial phase
A-9	Sine	5	30	Four different codes, zero initial phase
A-10	Sine	5	40	Four different codes, zero initial phase
A-11	Sine	5	60	Four different codes, zero initial phase
A-12	Circular	5	20	Five different values of initial phase, (1)-code
A-13	Circular	5	20	Five different values of initial phase, (1,2)-code
A-14	Circular	5	20	Five different values of initial phase, (1,3)-code
A-15	Circular	5	20	Five different values of initial phase, (1,2,3)-code
A-16	Circular	7.5	10	Four different codes, zero initial phase
A-17	Circular	5	20	Four different codes, zero initial phase
A-18	Circular	7.5	30	Four different codes, zero initial phase
A-19	Circular	10	40	Four different codes, zero initial phase
A-20	Circular	12.5	50	Four different codes, zero initial phase
A-21	Both	5	20	(1)-code, zero initial phase
A-22	Both	5	20	(1,2)-code, zero initial phase
A-23	Both	5	20	(1,3)-code, zero initial phase
A-24	Both	5	20	(1,2,3)-code, zero initial phase

Figure A-1 (a)

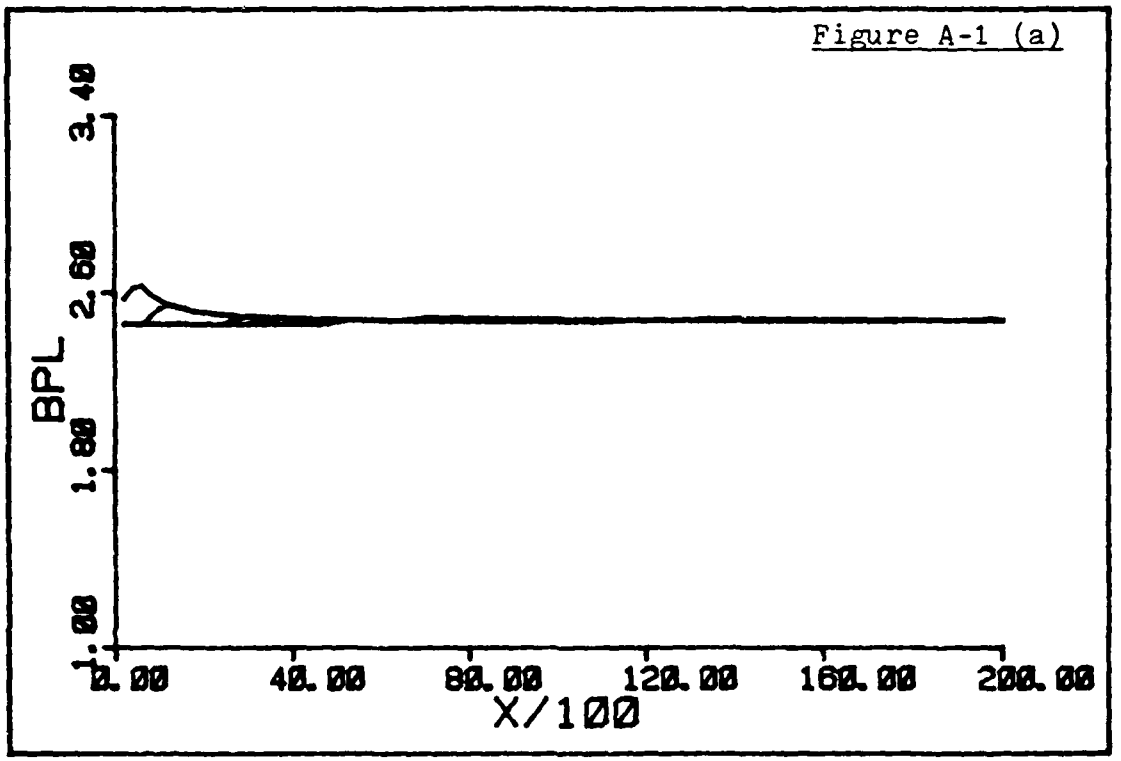


Figure A-1 (b)

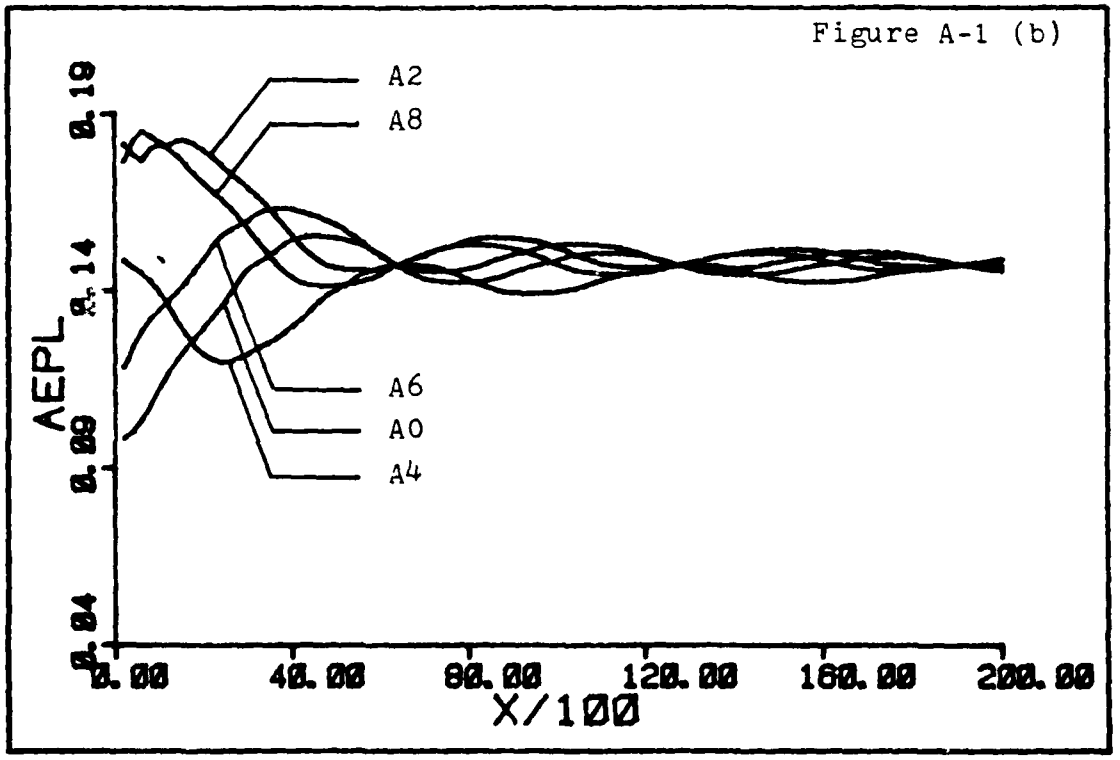


Figure A-2 (a)

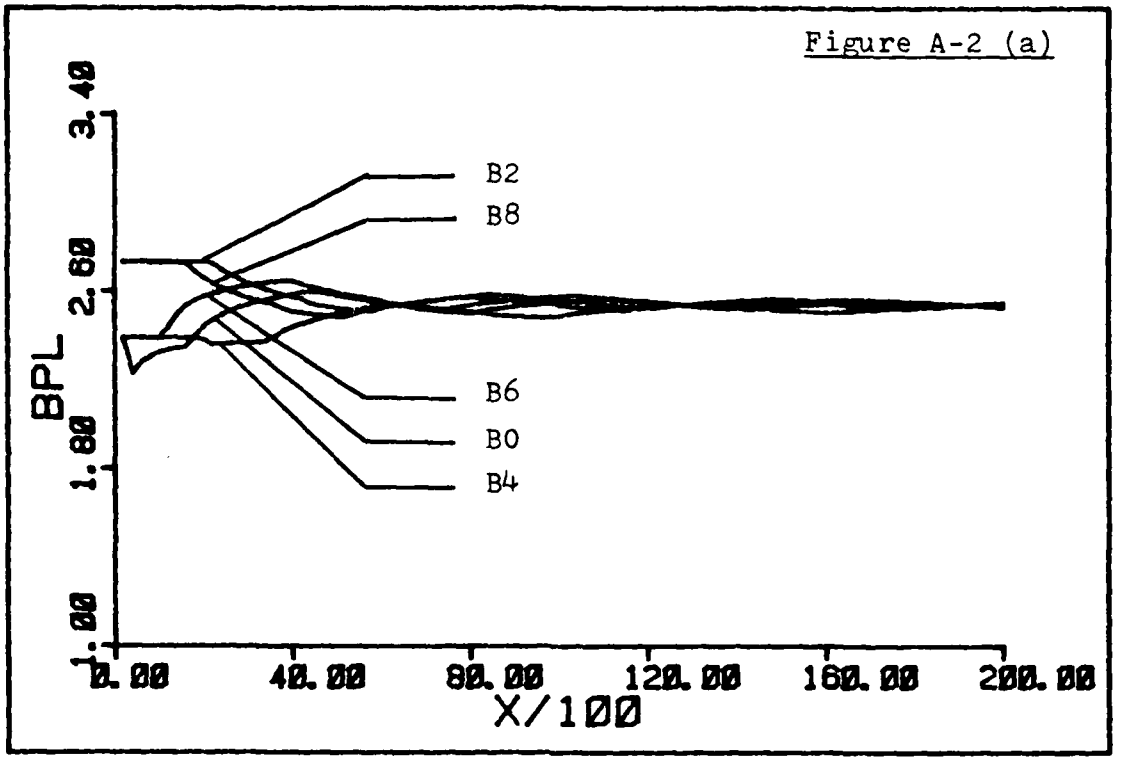


Figure A-2 (b)

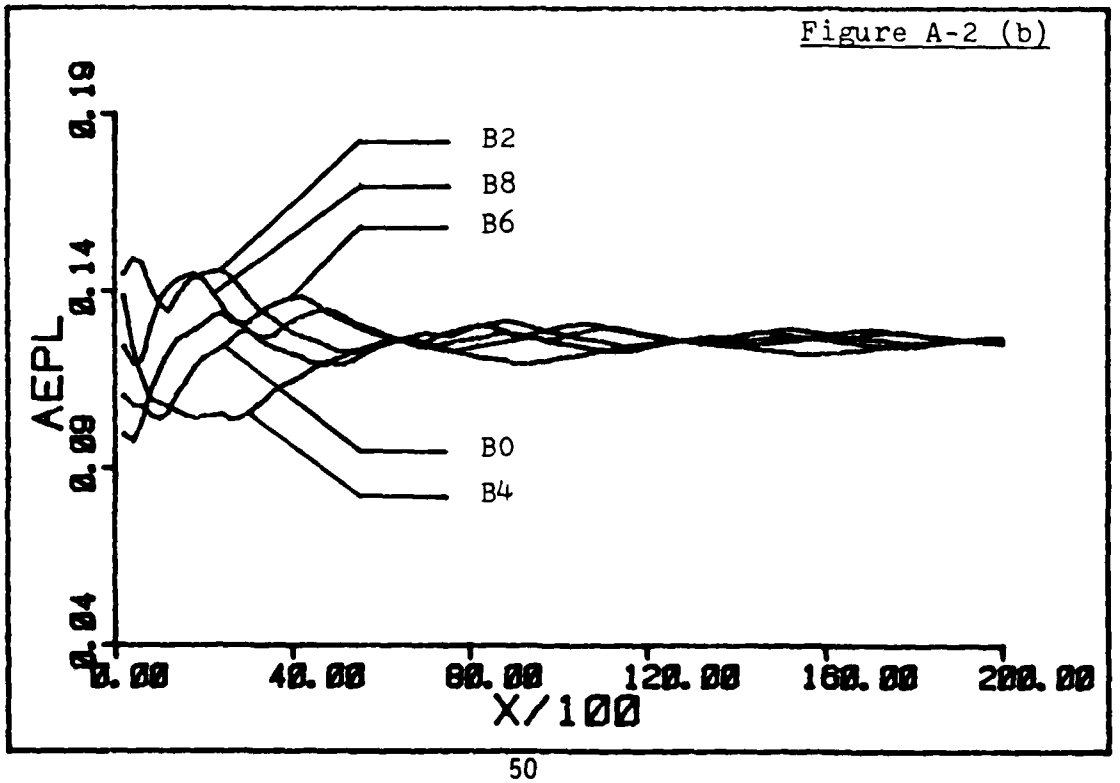


Figure A-3 (a)

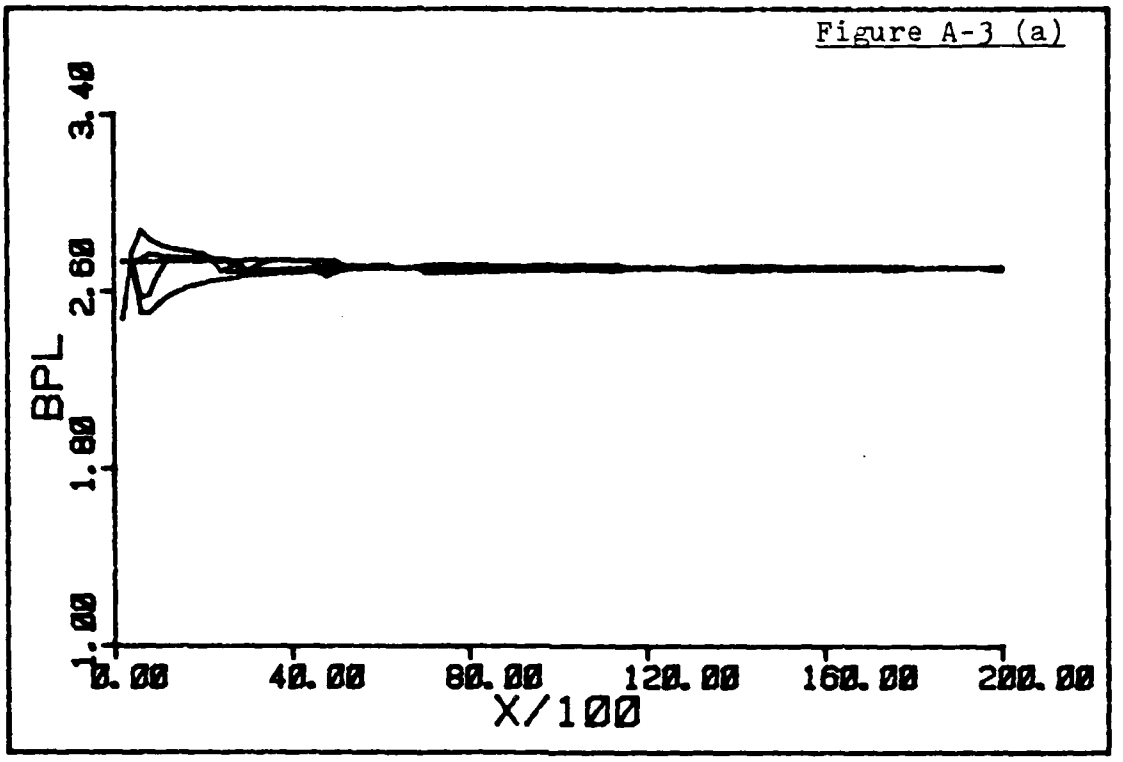


Figure A-3 (b)

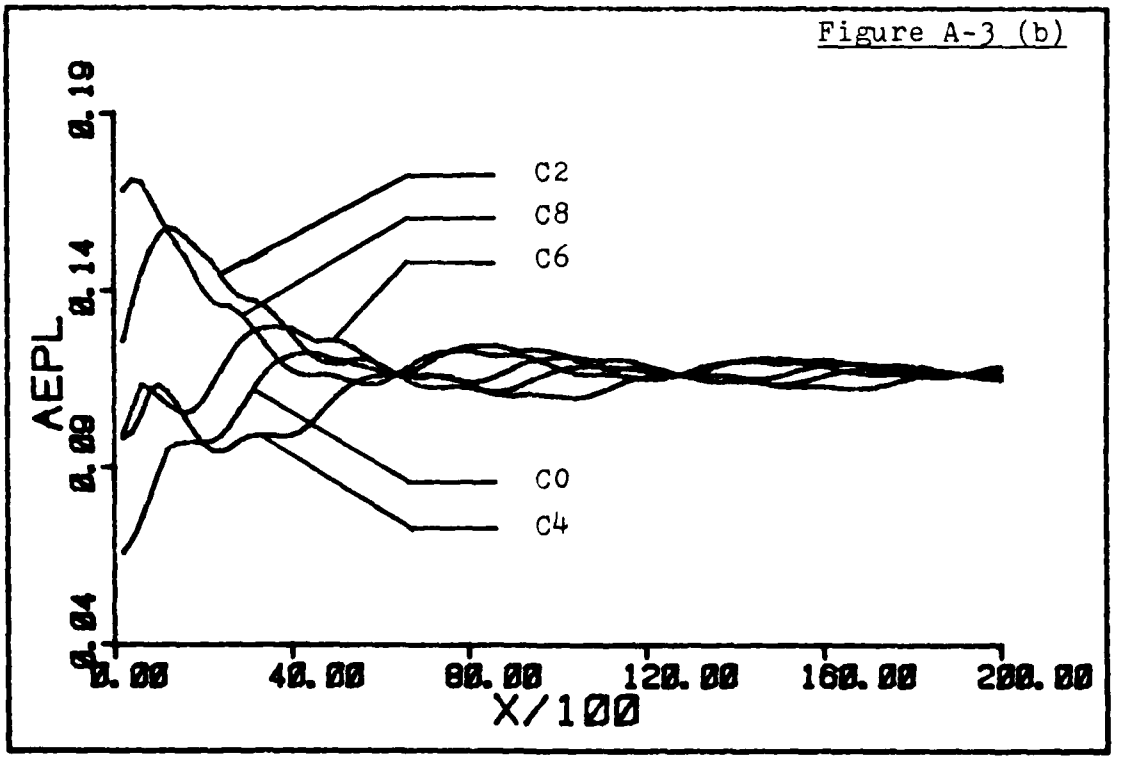


Figure A-4 (a)

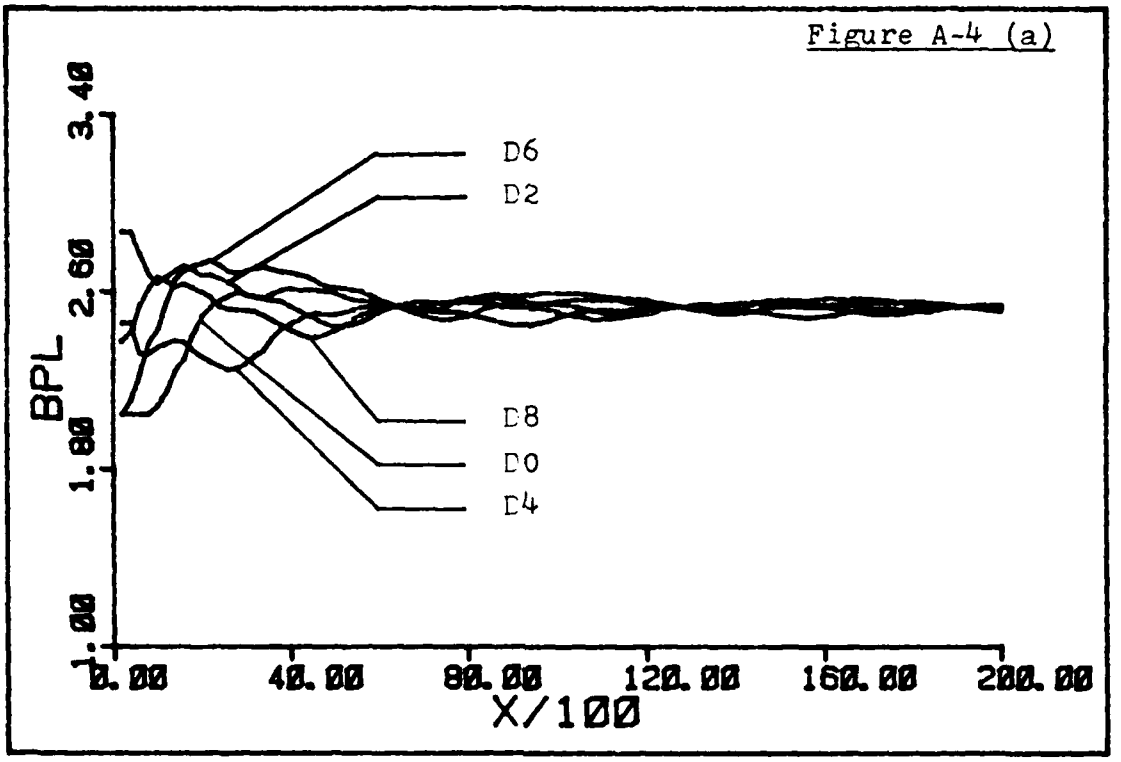


Figure A-4 (b)

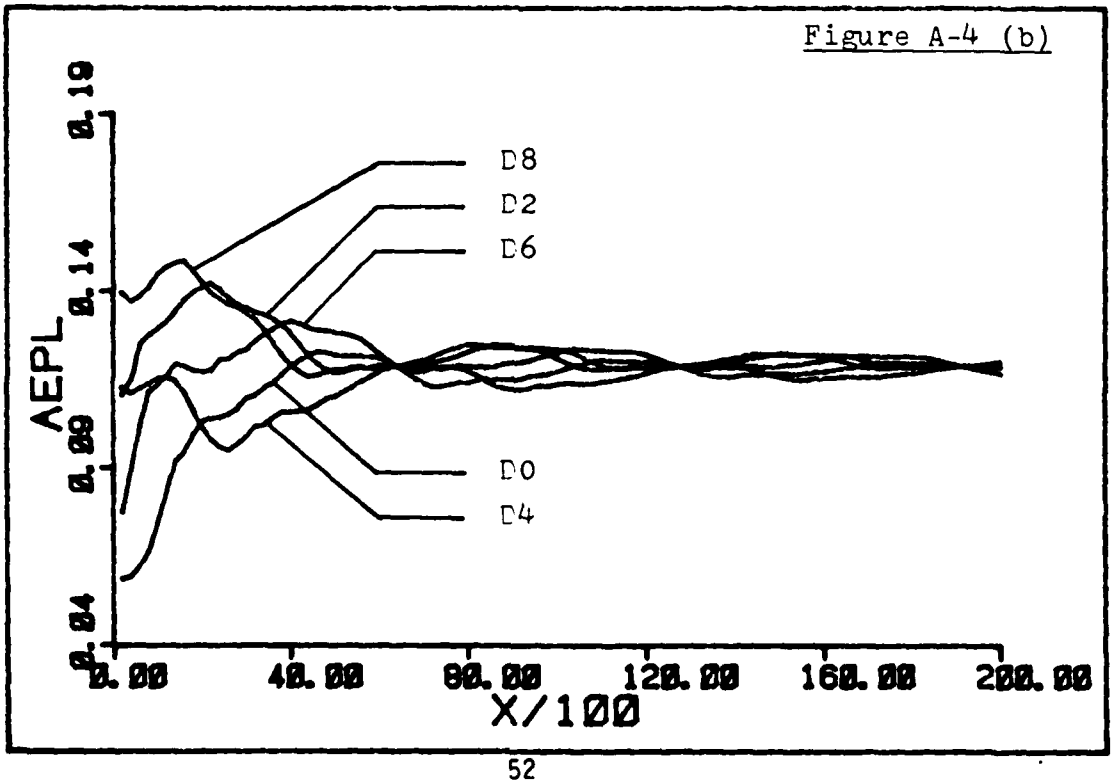


Figure A-5 (a)

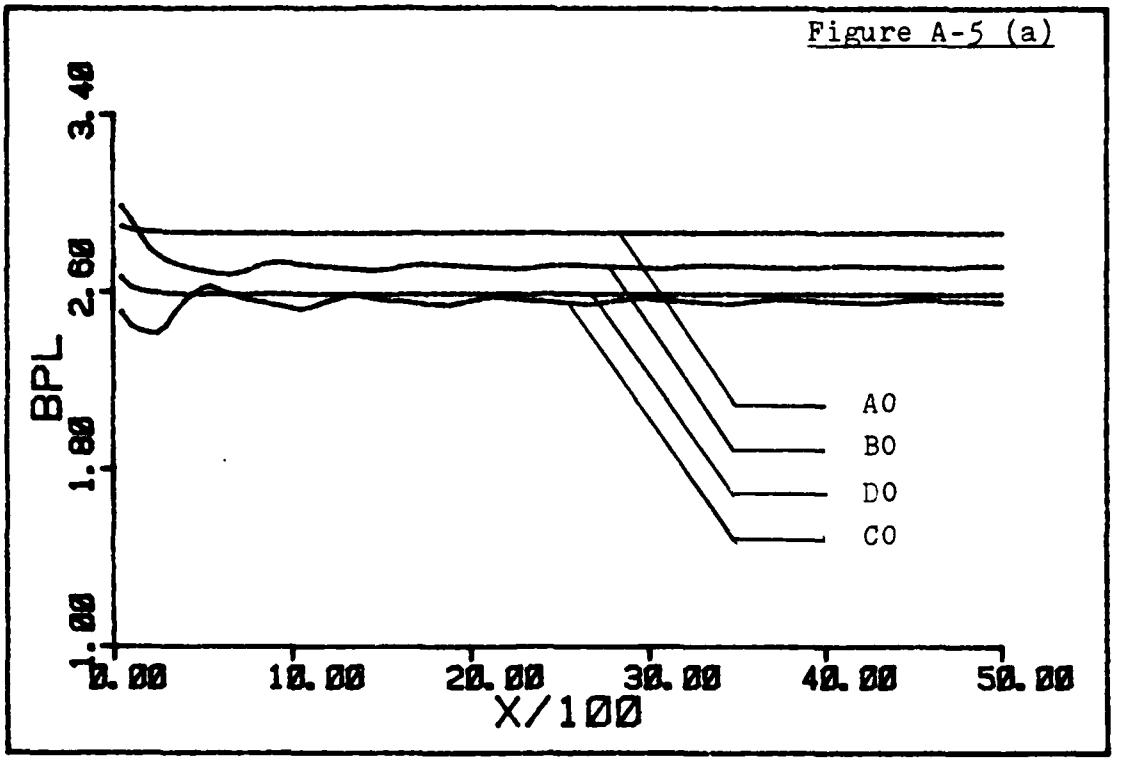


Figure A-5 (b)

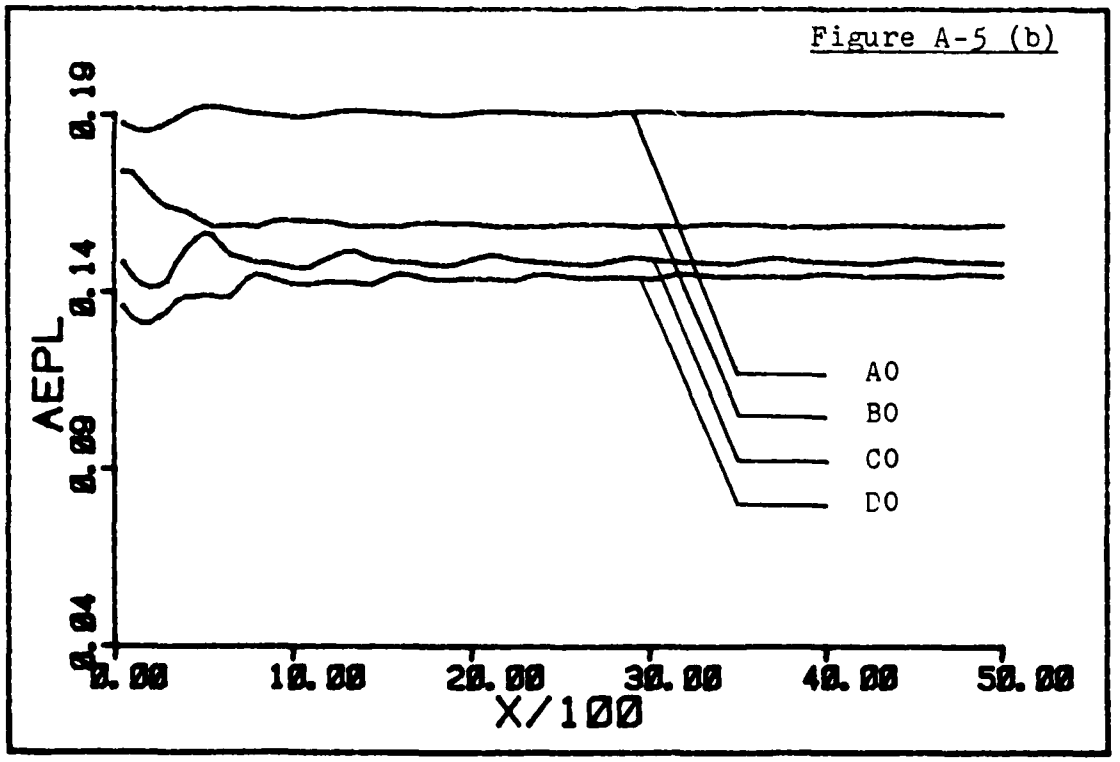


Figure A-6 (a)

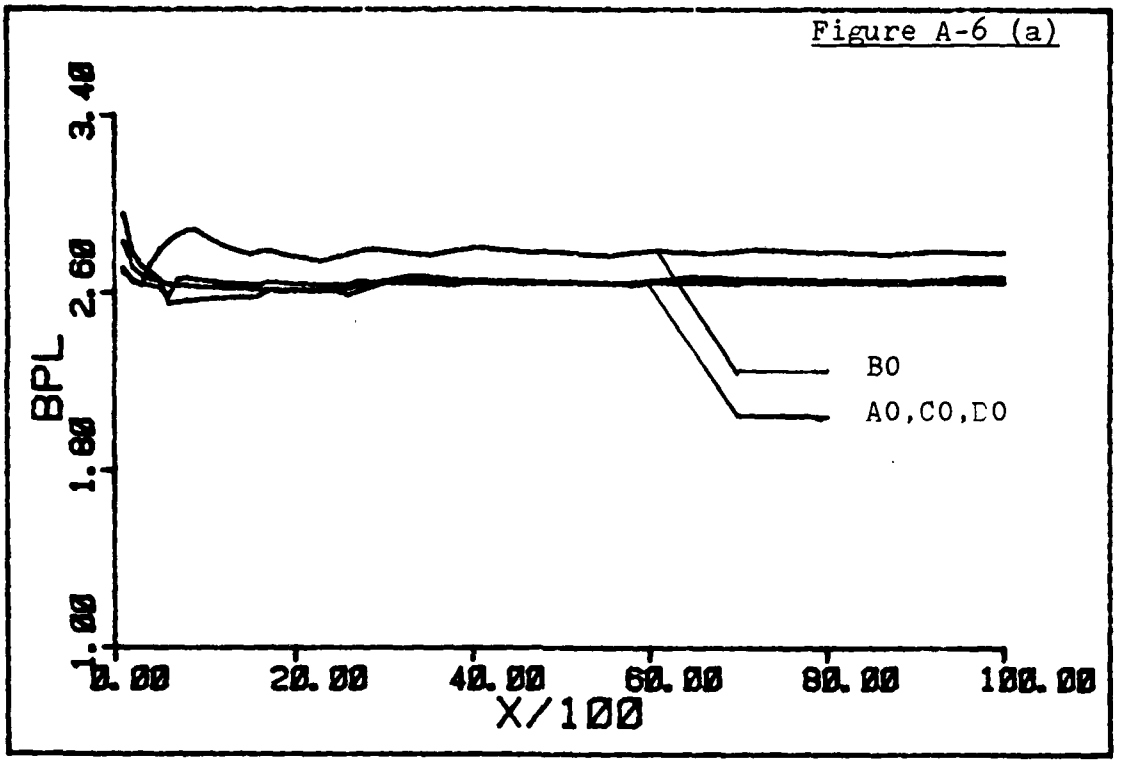


Figure A-6 (b)

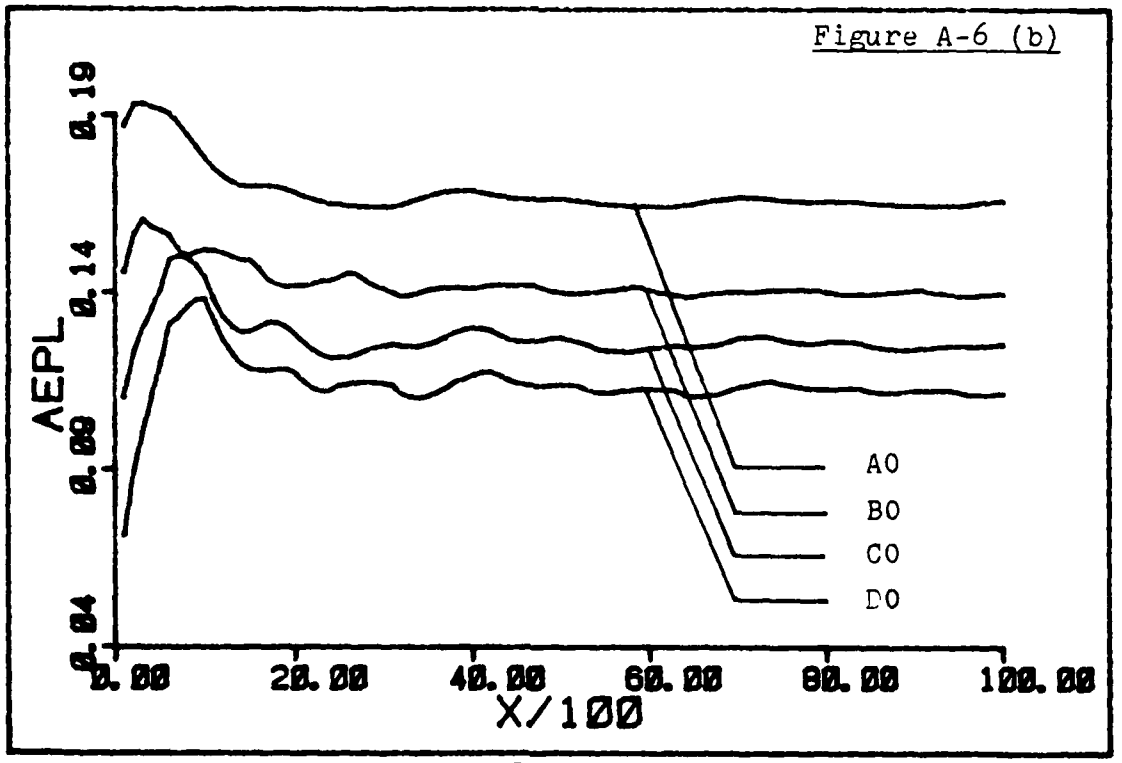


Figure A-7 (a)

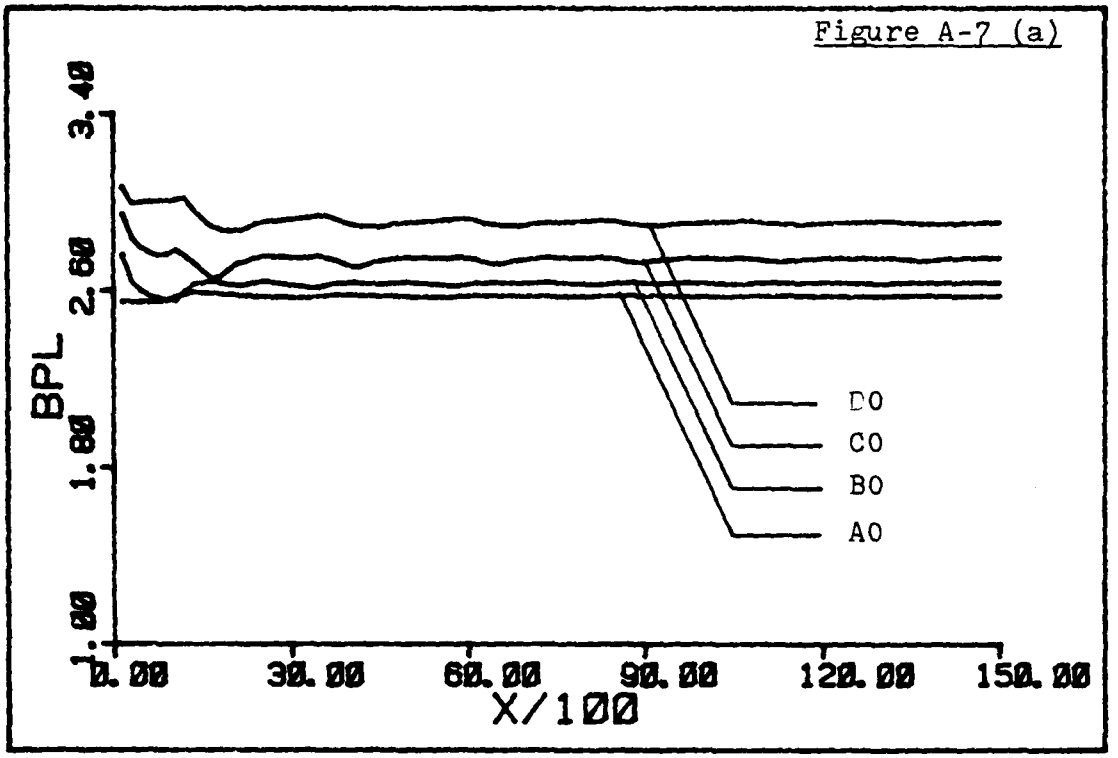


Figure A-7 (b)

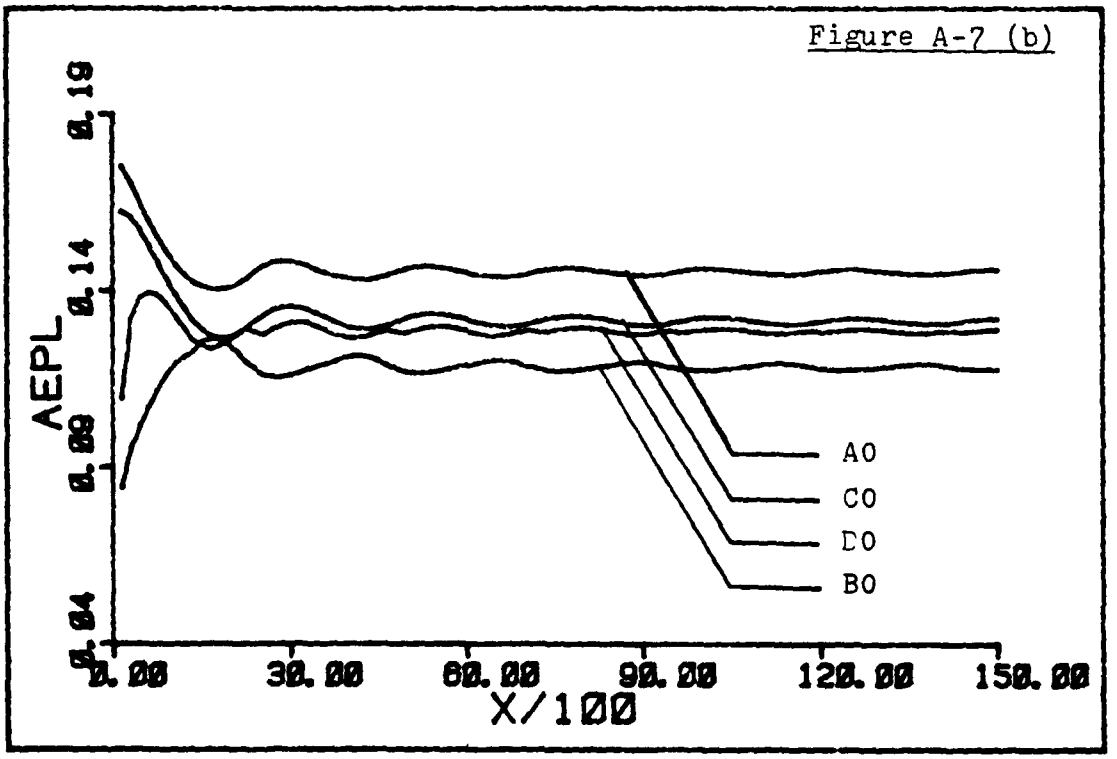


Figure A-8 (a)

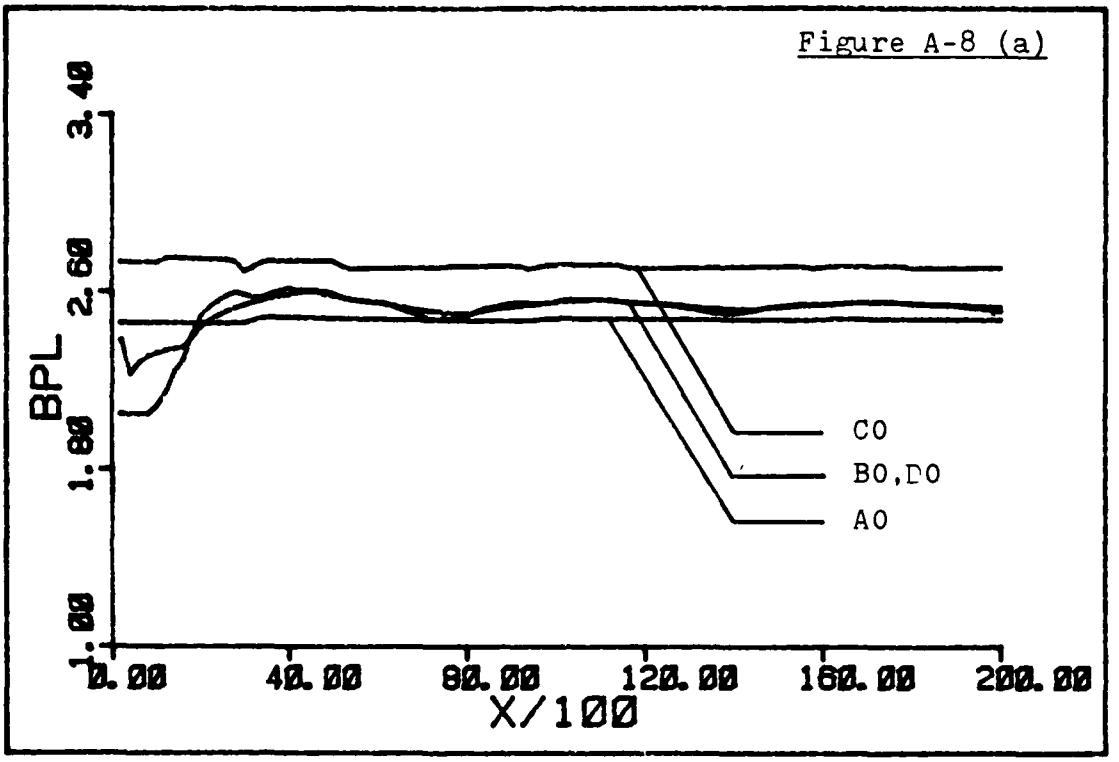


Figure A-8 (b)

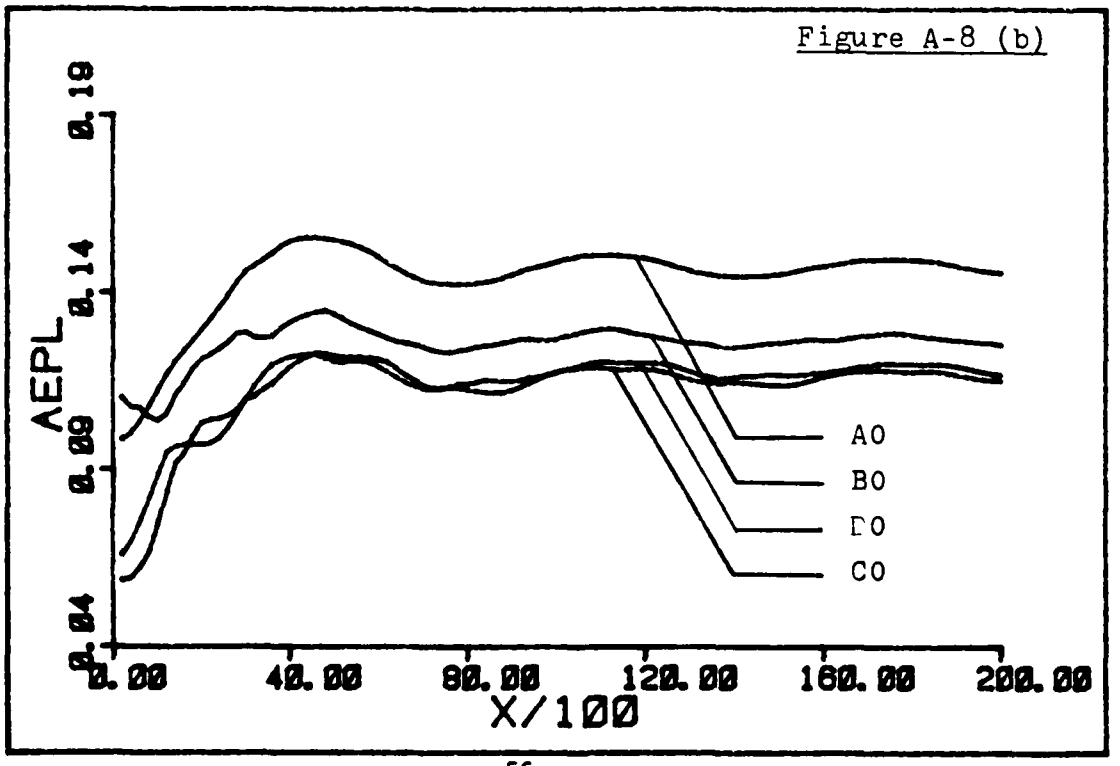


Figure A-9 (a)

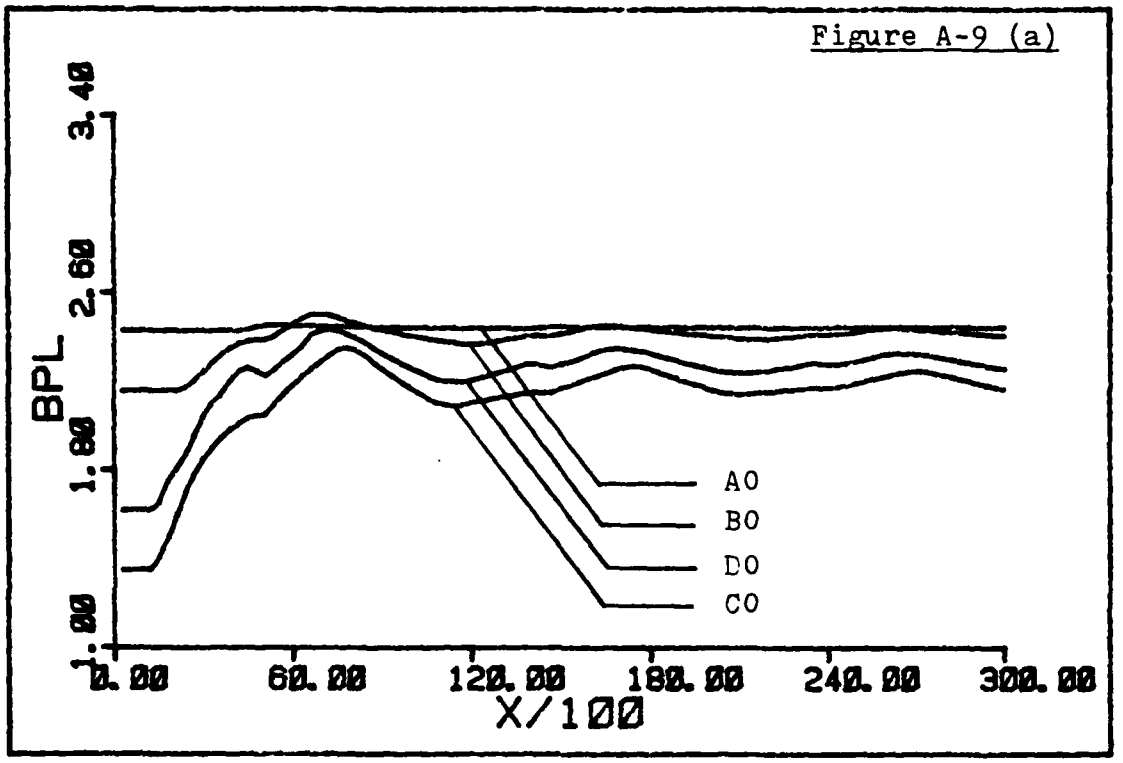


Figure A-9 (b)

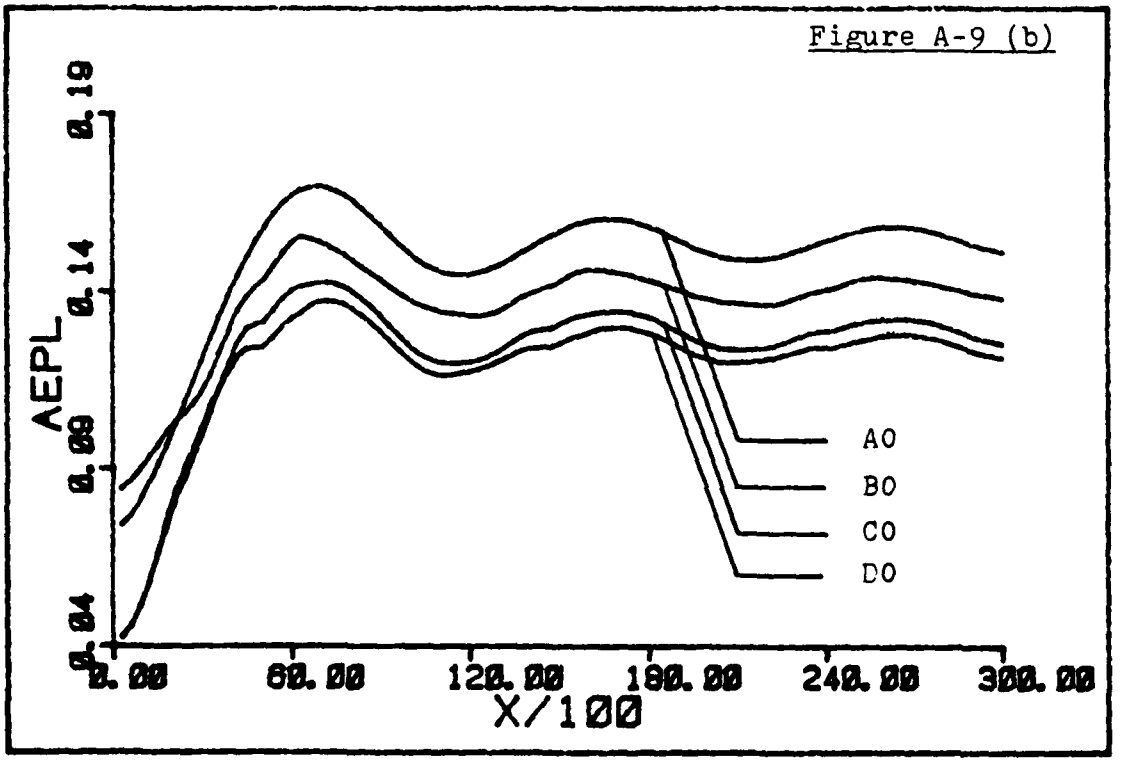


Figure A-10 (a)

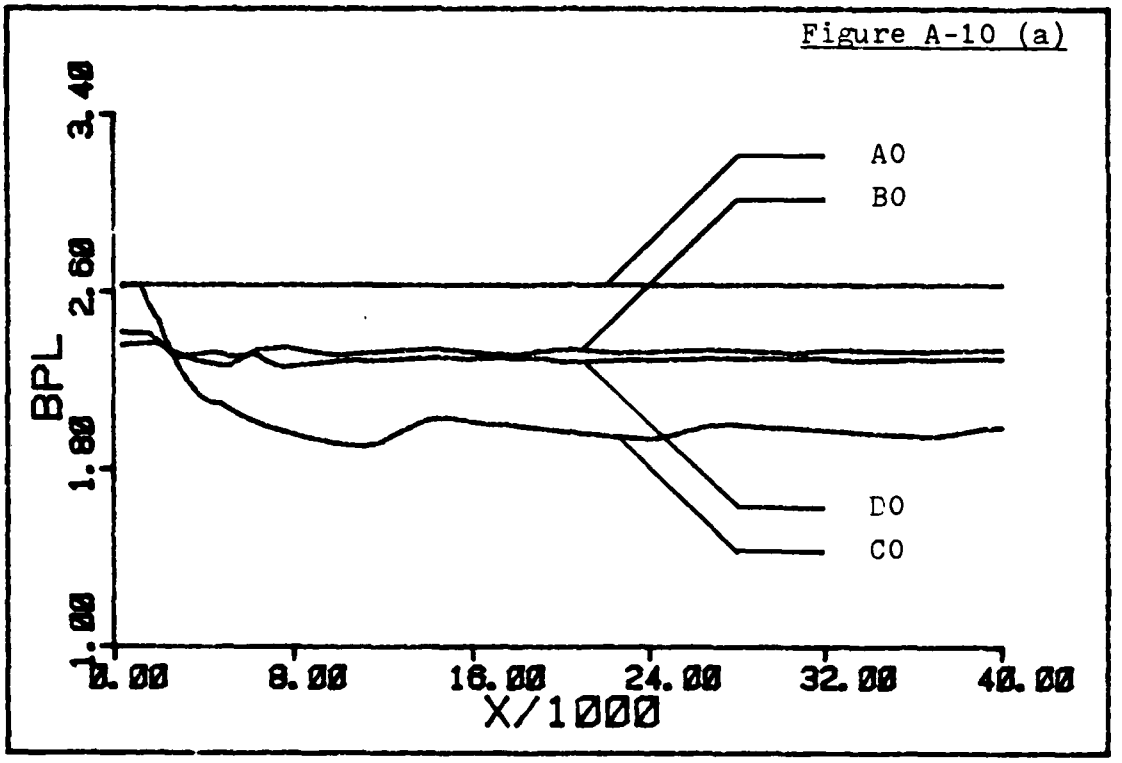


Figure A-10 (b)

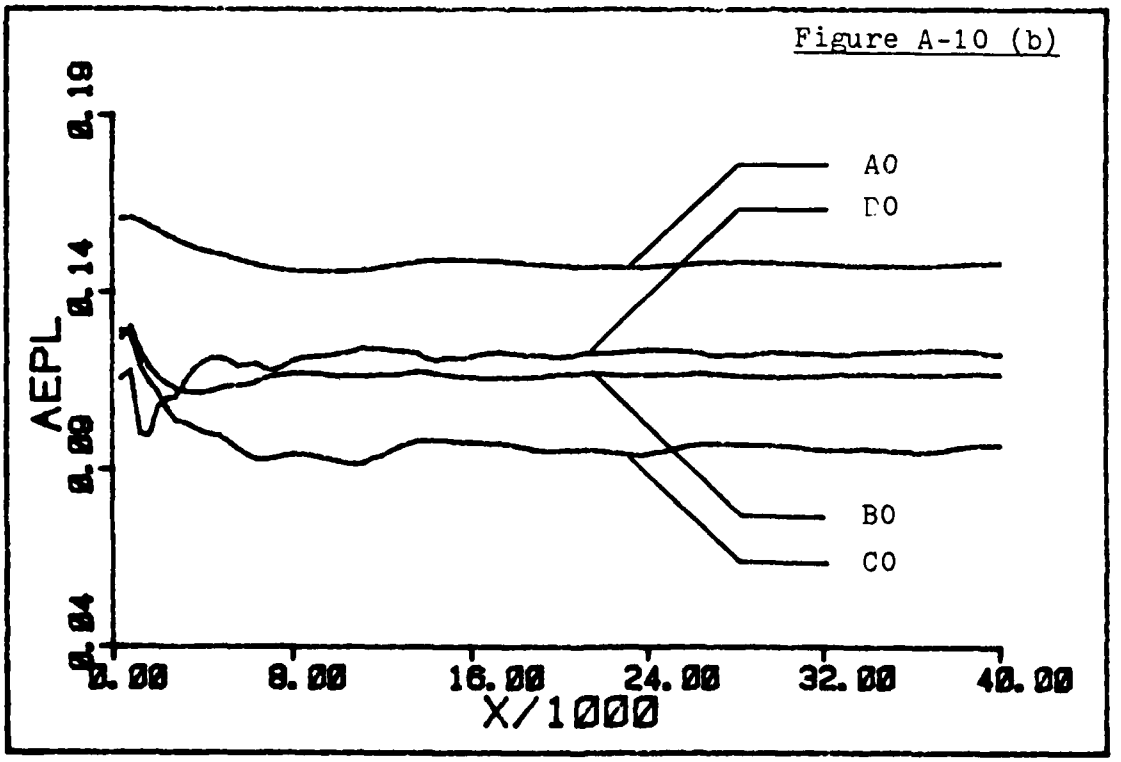


Figure A-12 (a)

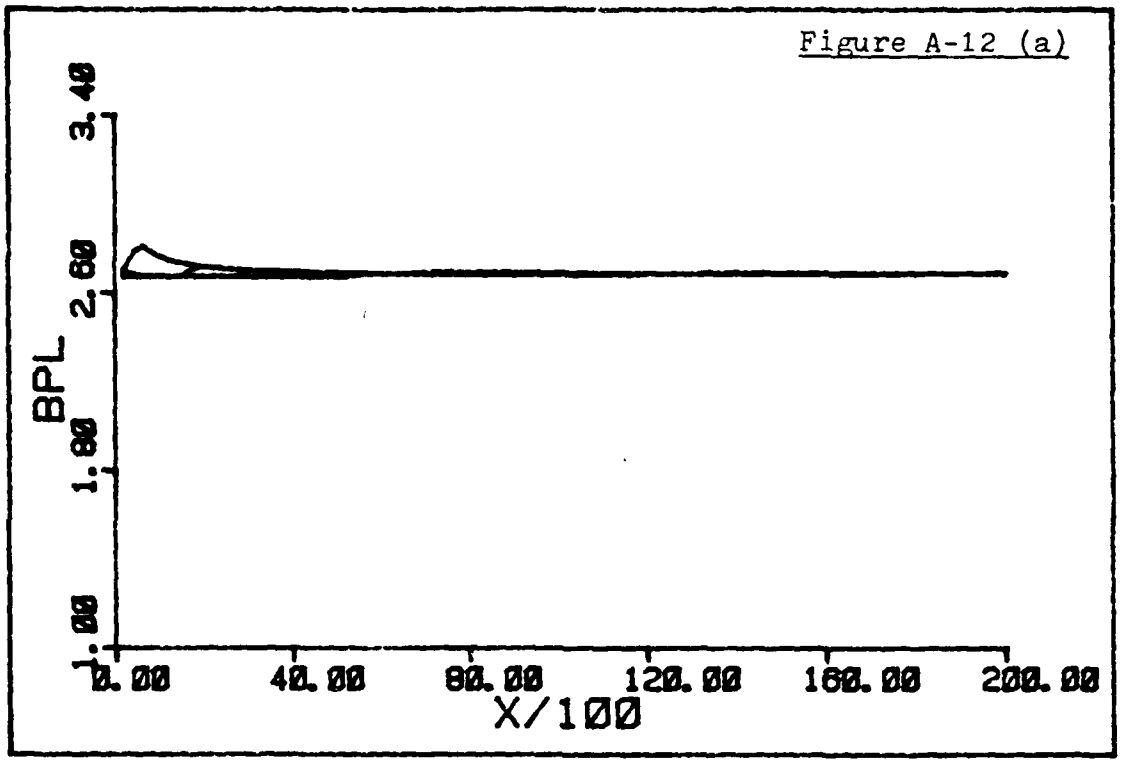


Figure A-12 (b)

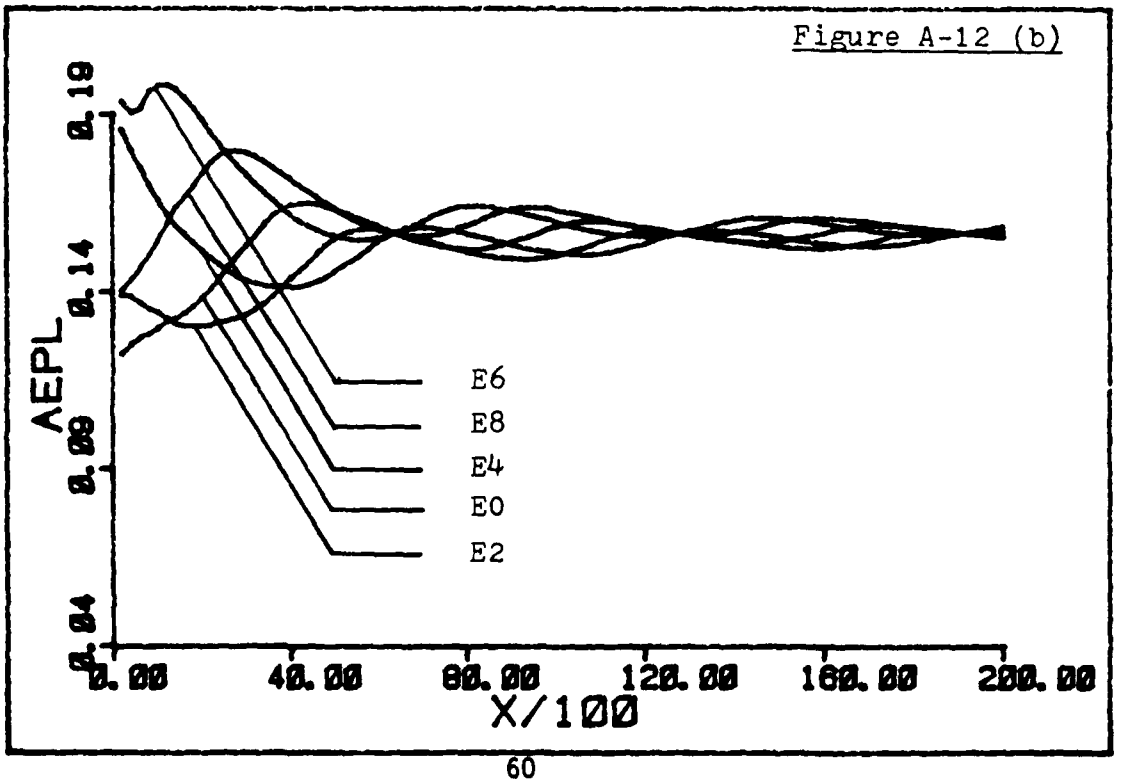


Figure A-11 (a)

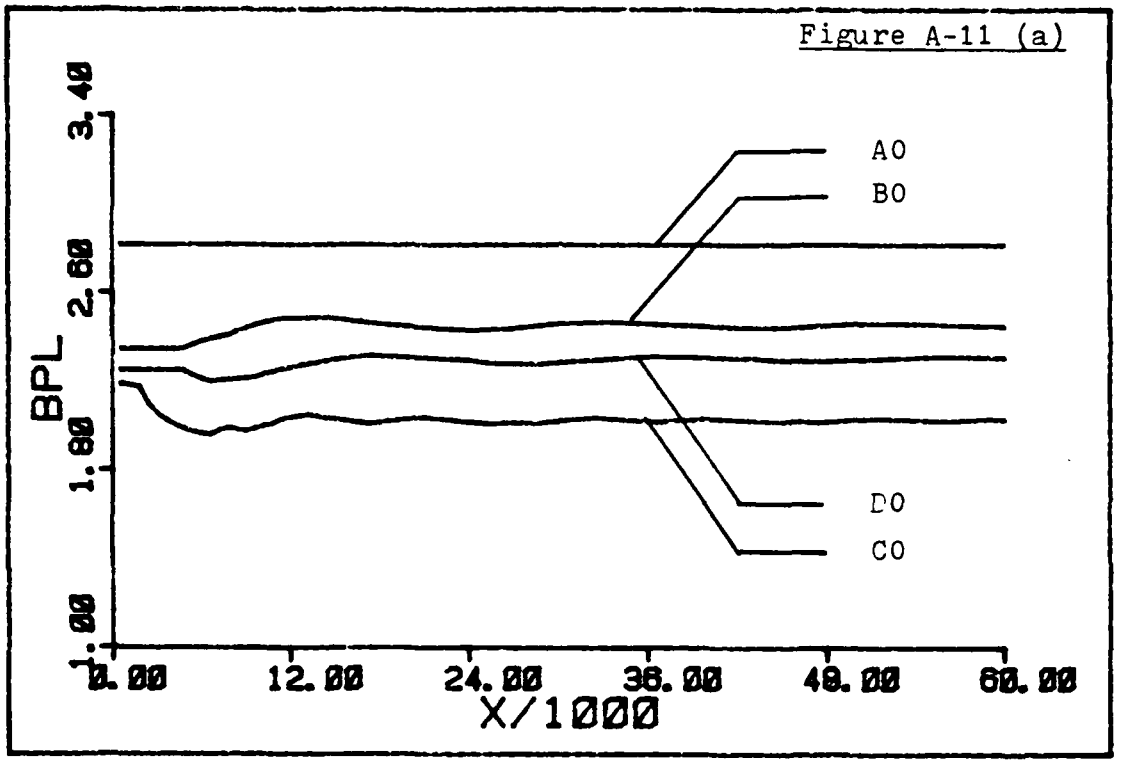


Figure A-11 (b)

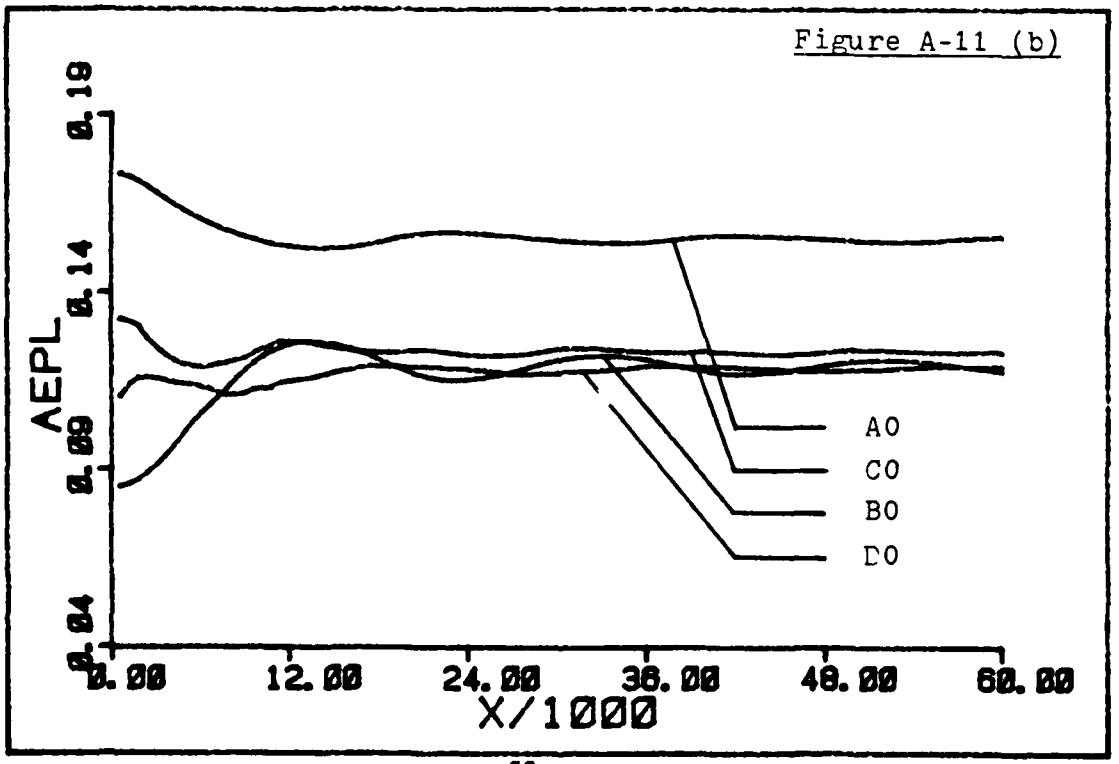


Figure A-13 (a)

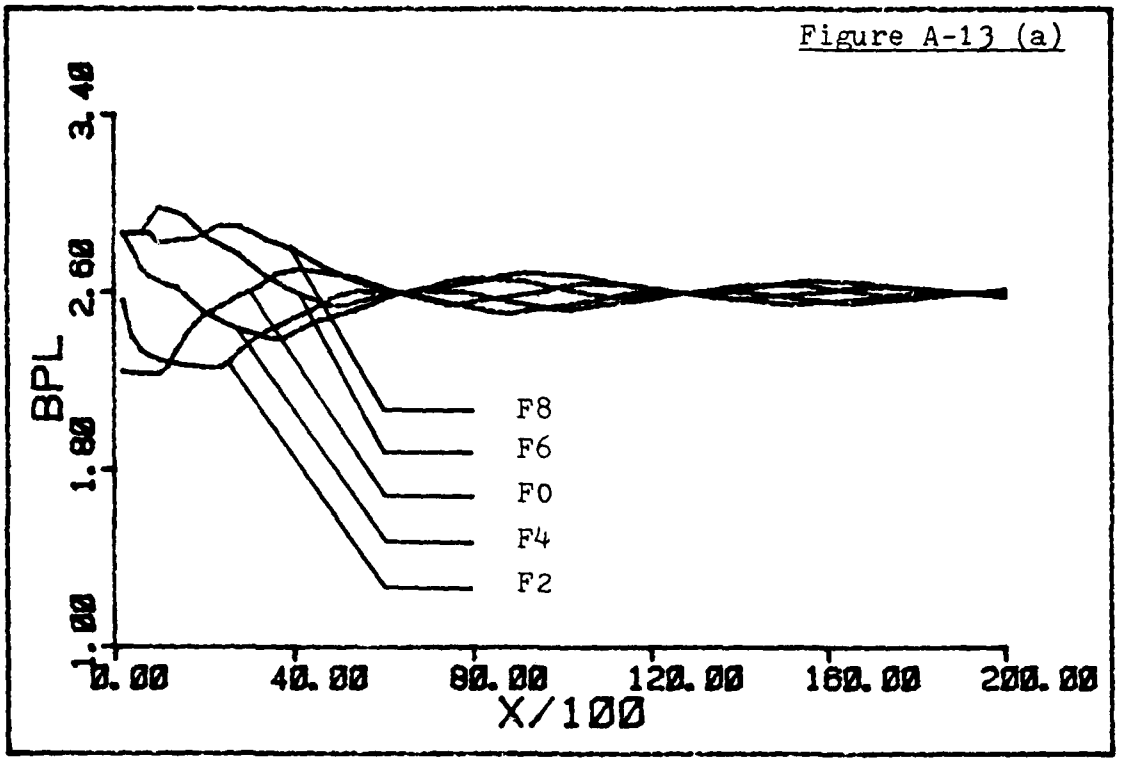


Figure A-13 (b)

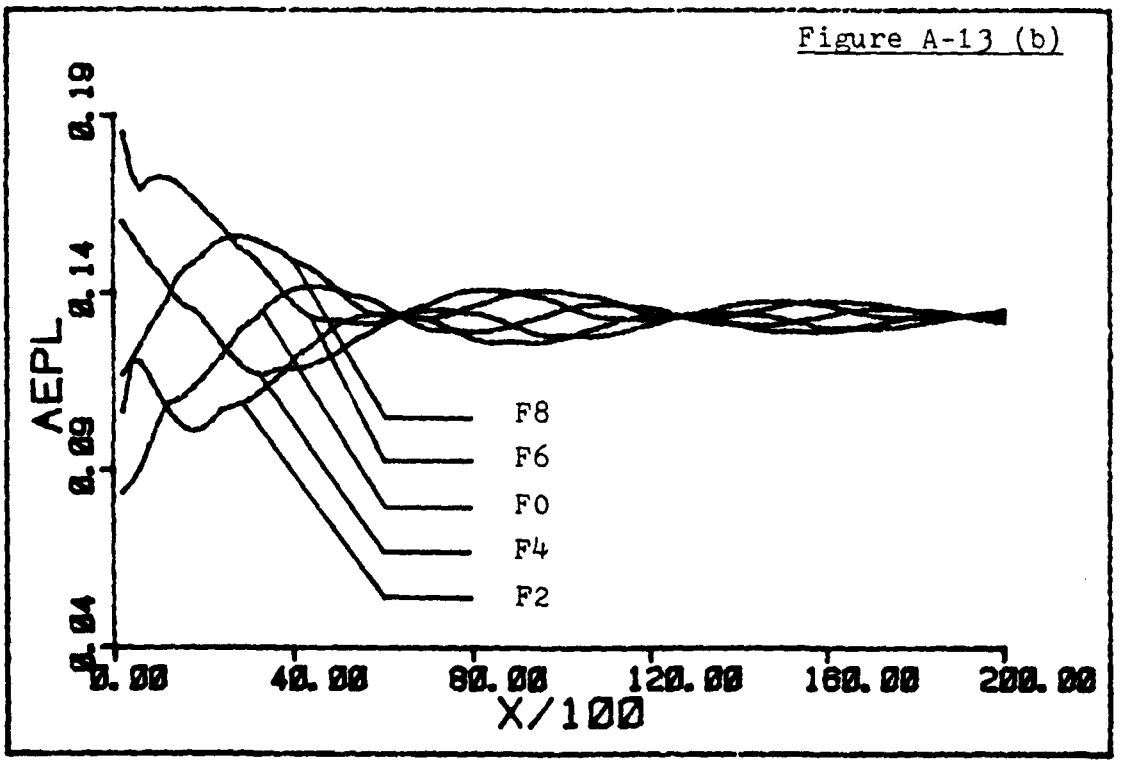


Figure A-14 (a)

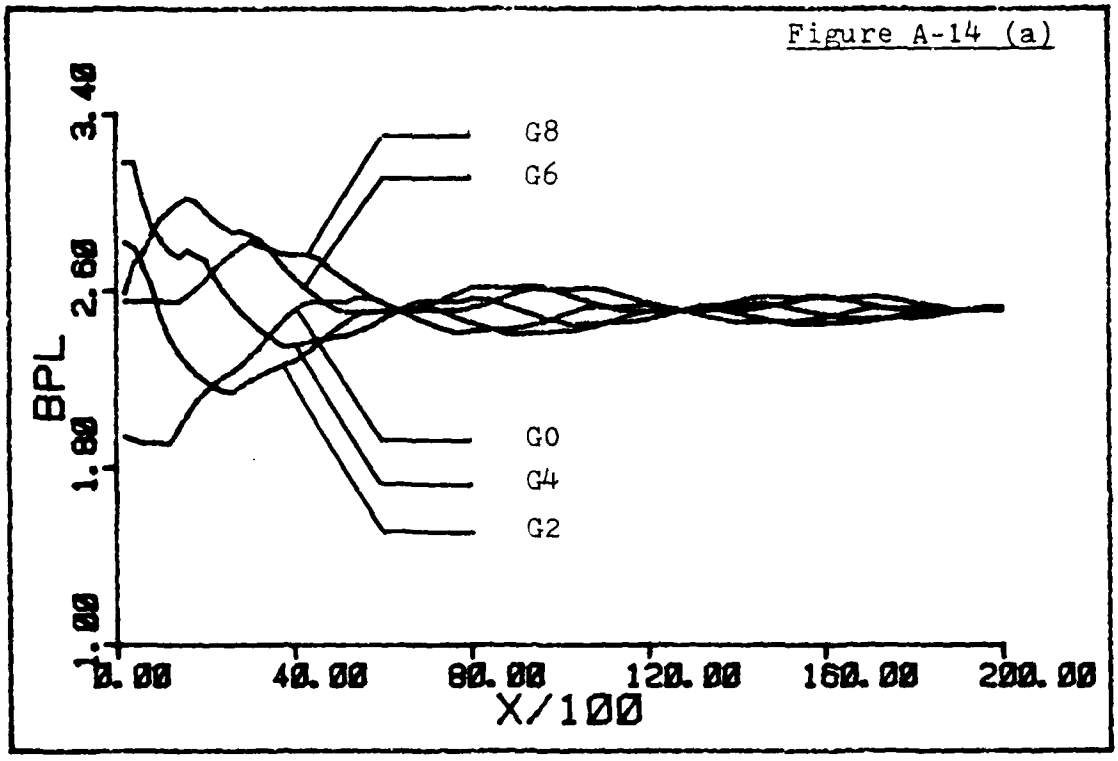


Figure A-14 (b)

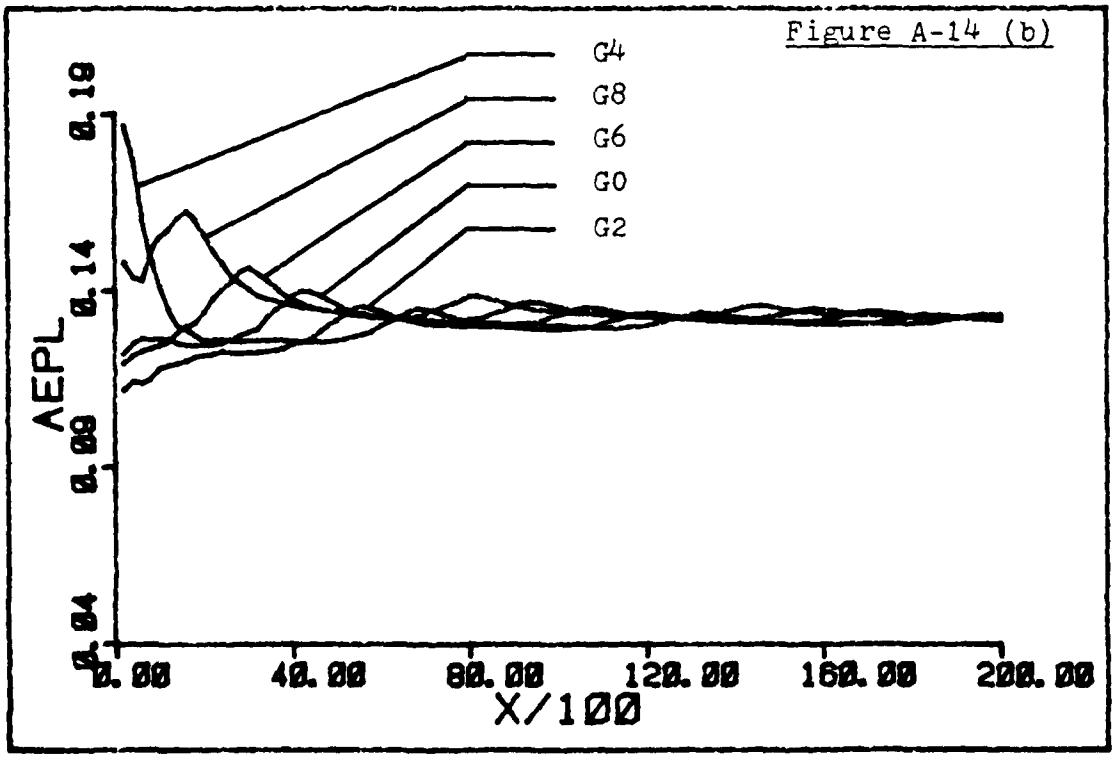


Figure A-15 (a)

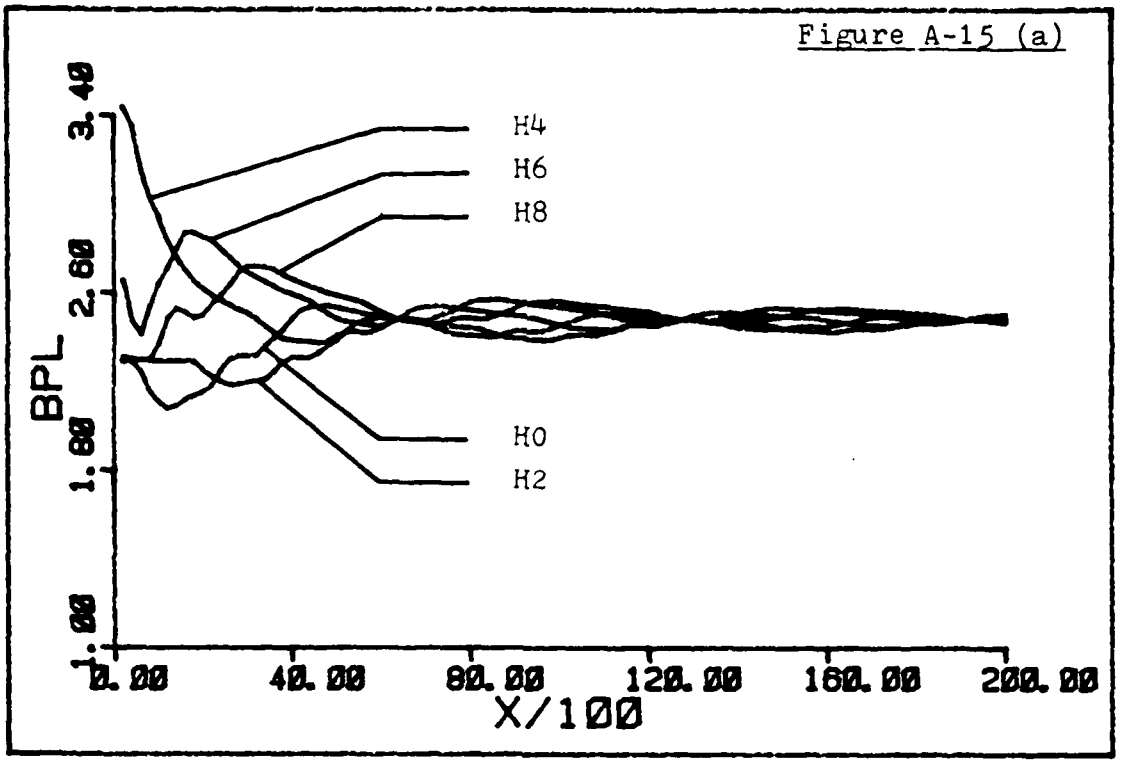


Figure A-15 (b)

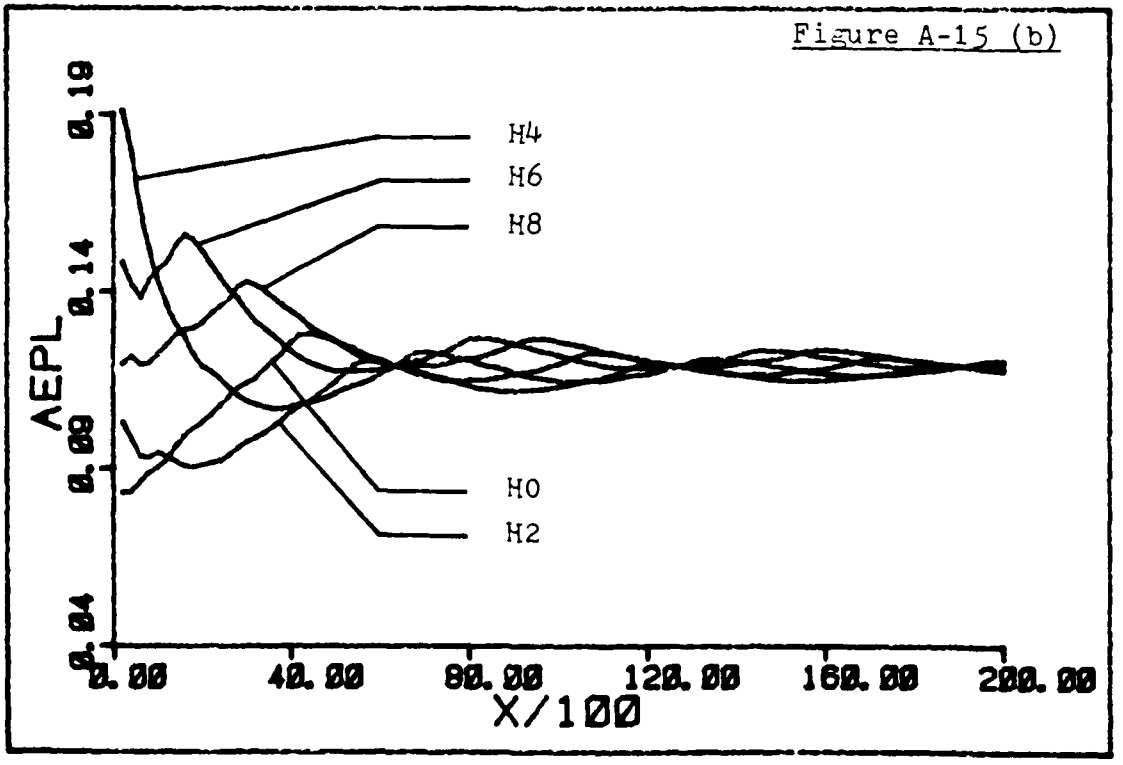


Figure A-16 (a)

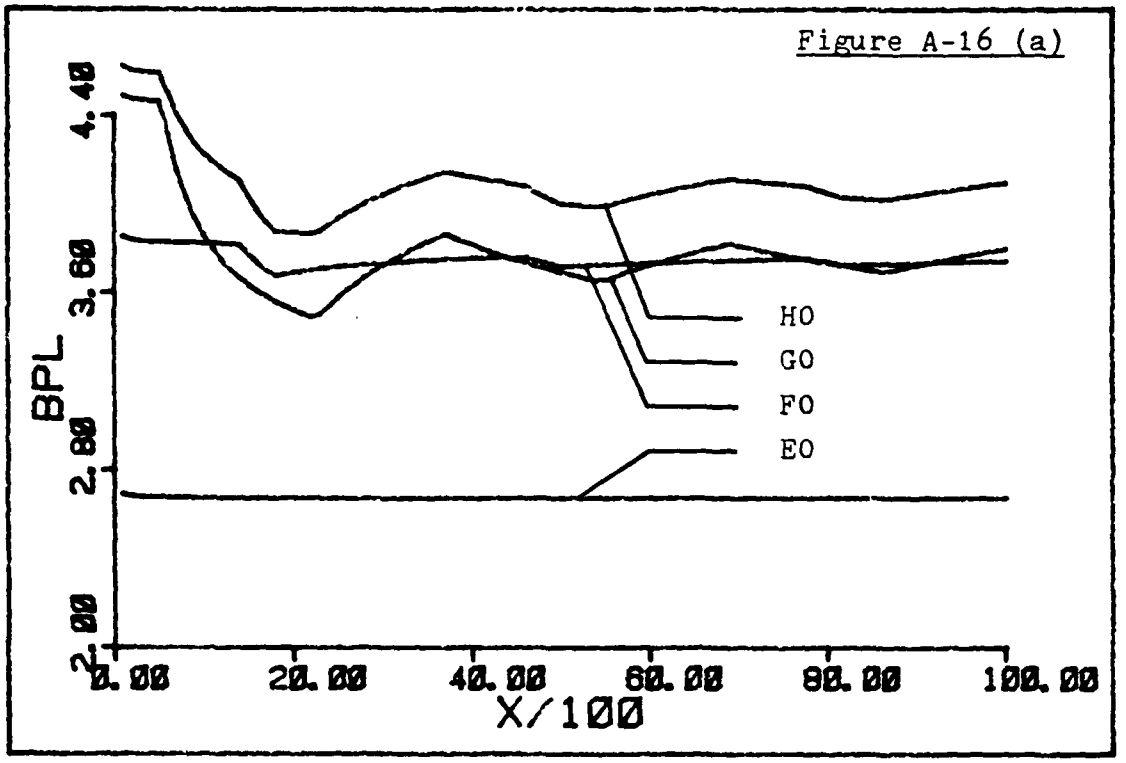


Figure A-16 (b)

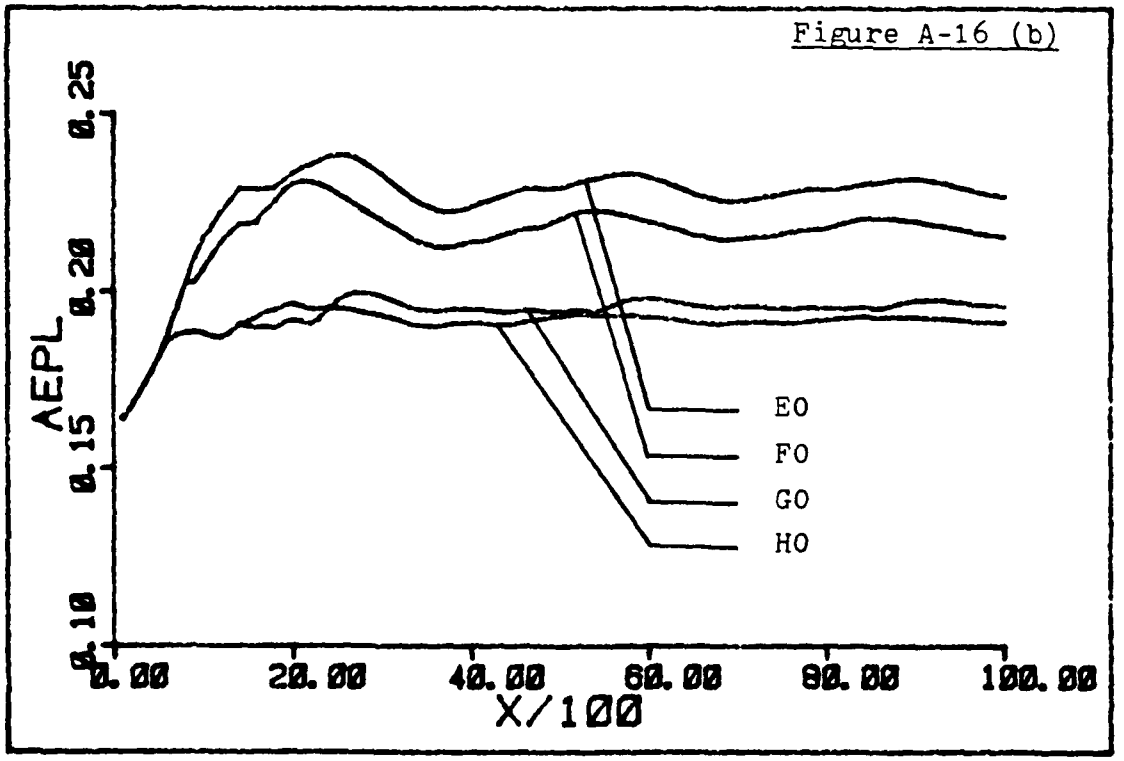


Figure A-17 (a)

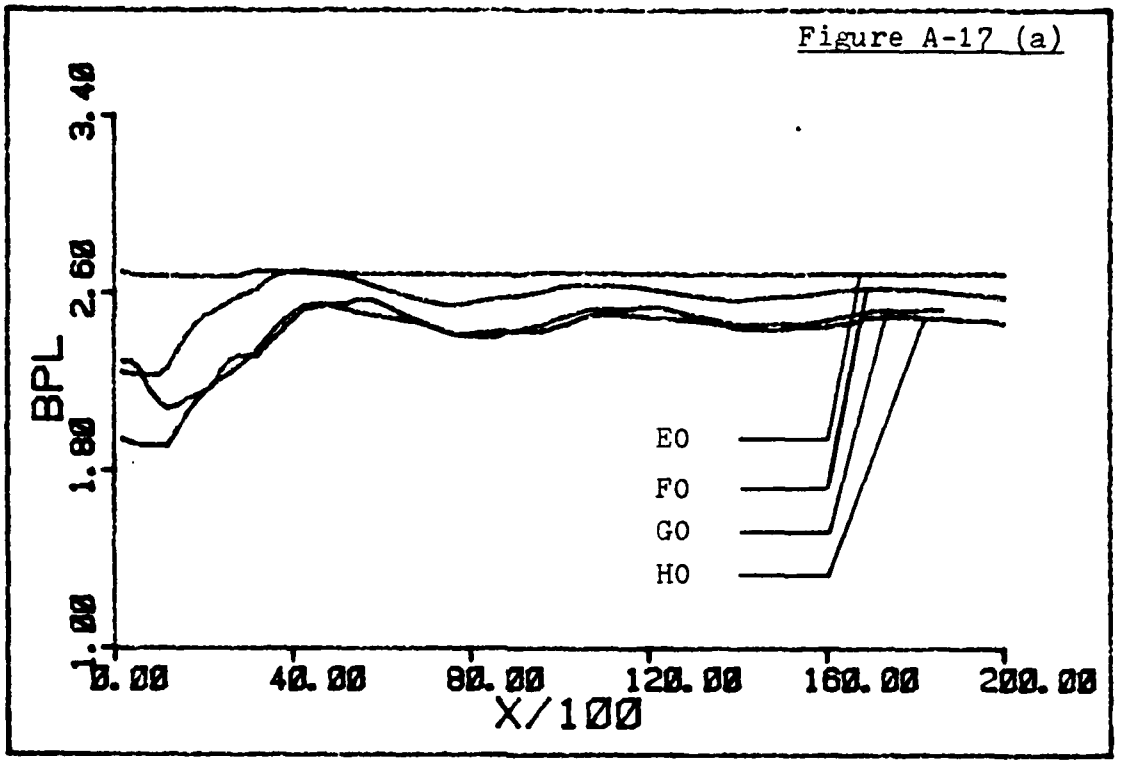


Figure A-17 (b)

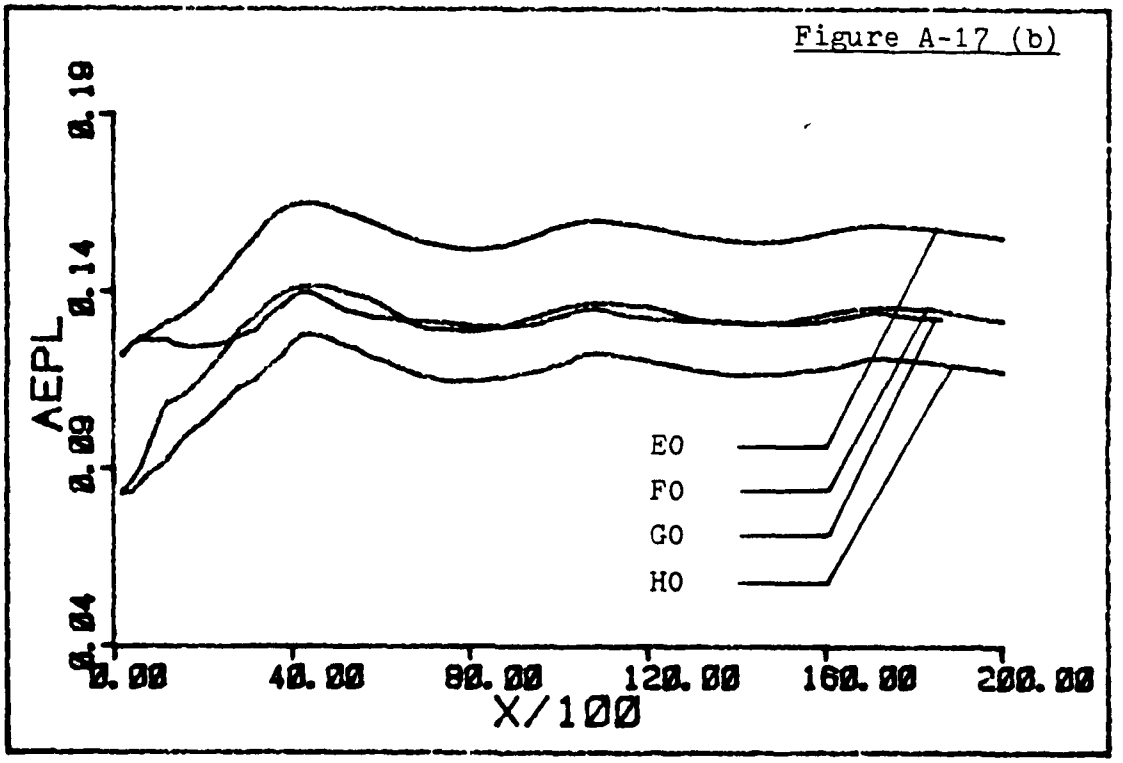


Figure A-18 (a)

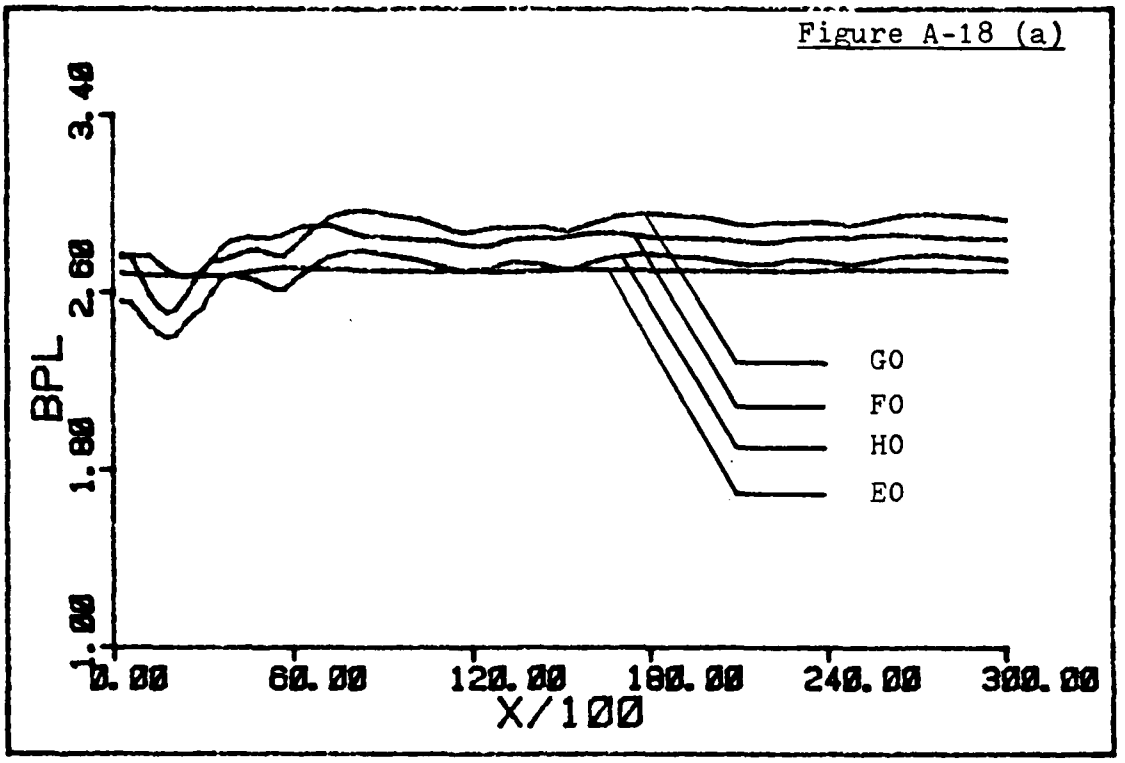


Figure A-18 (b)

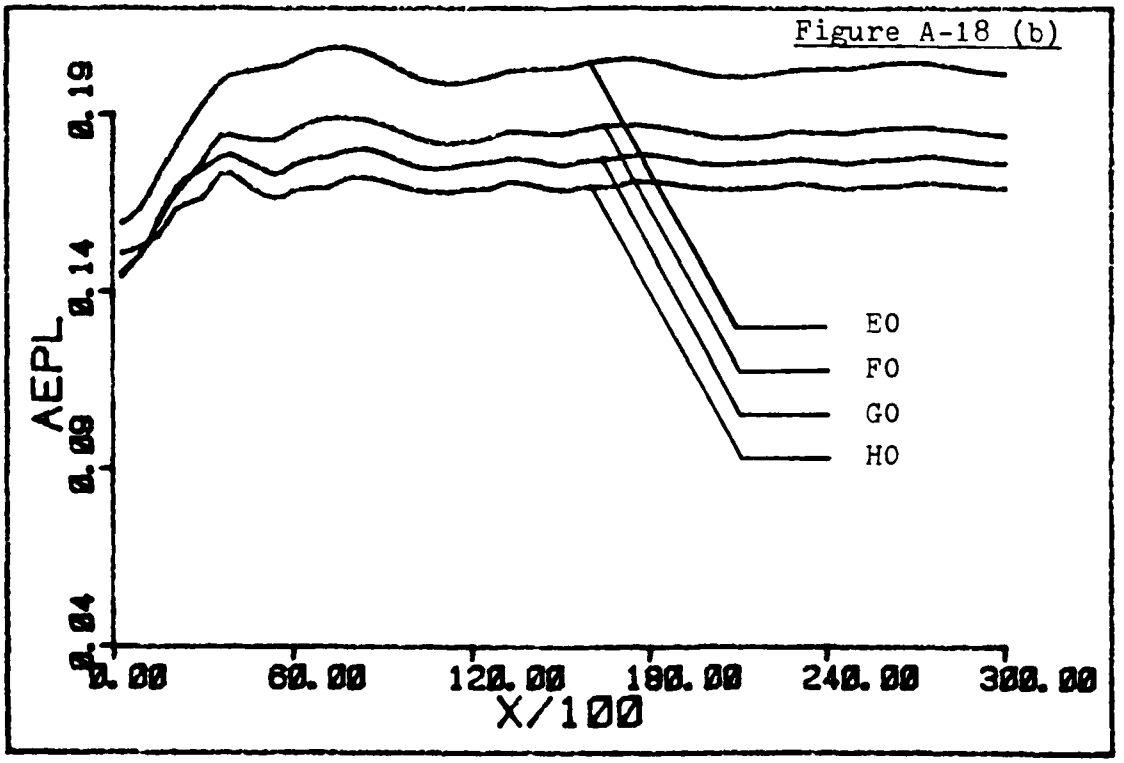


Figure A-19 (a)

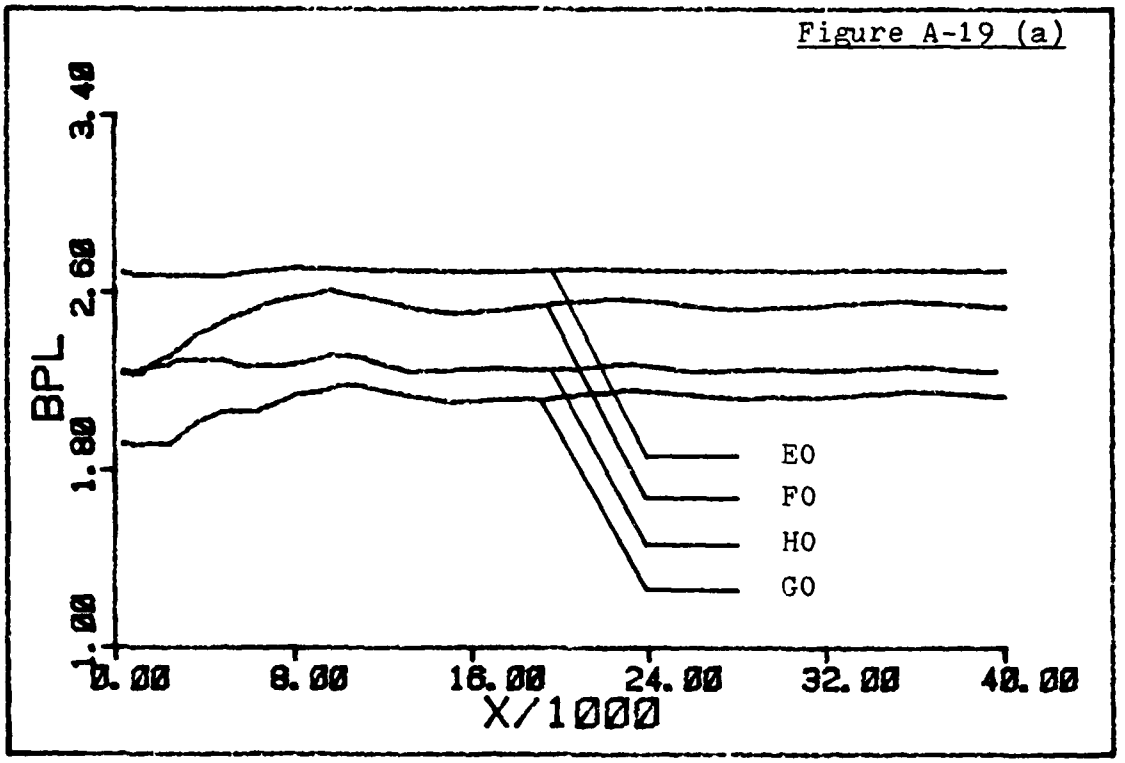


Figure A-19 (b)

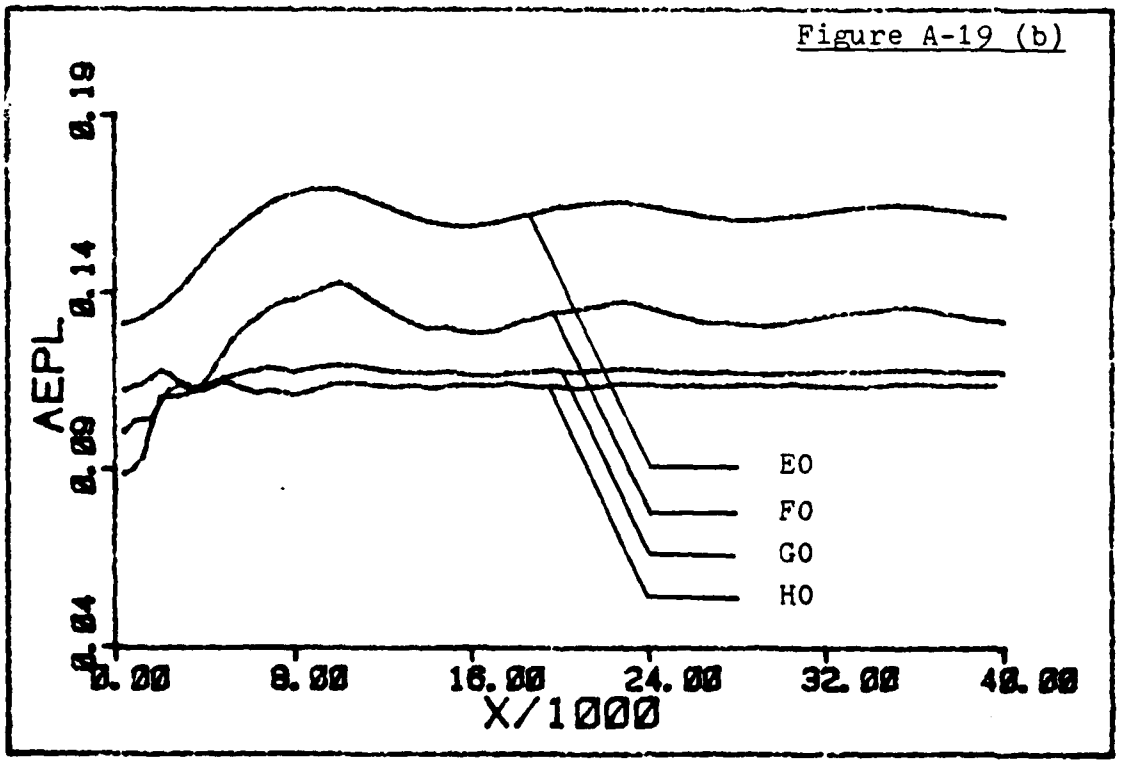


Figure A-20 (a)

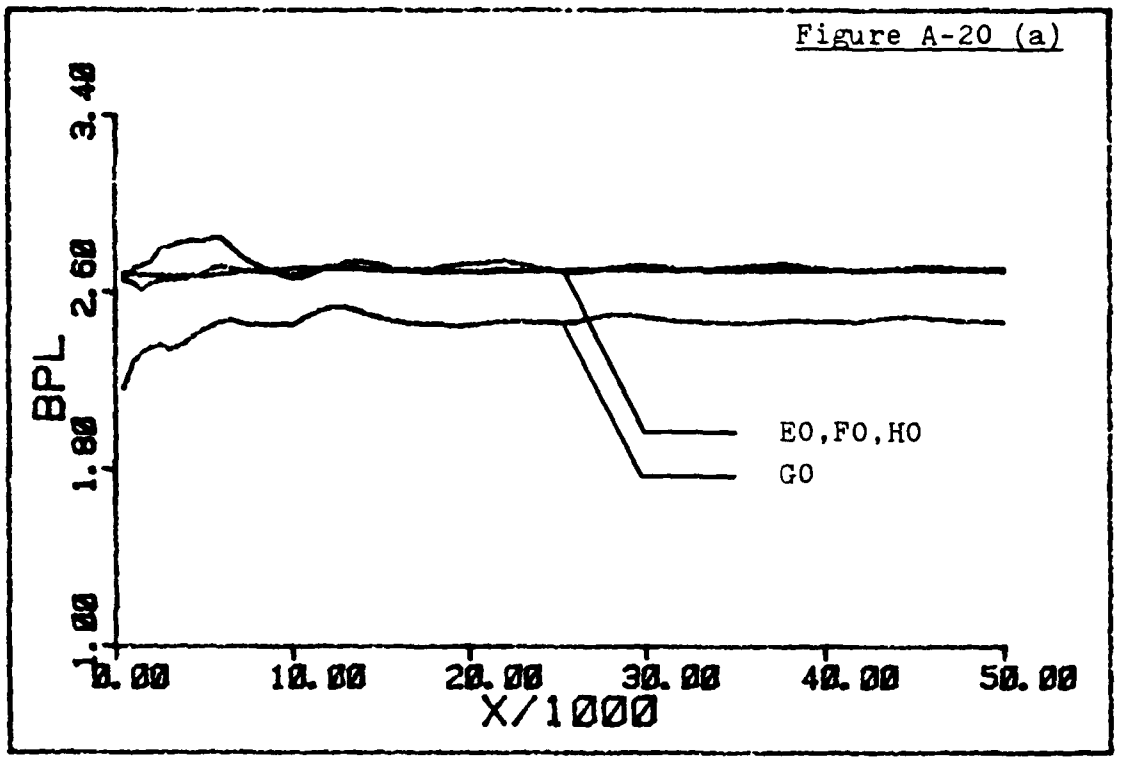


Figure A-20 (b)

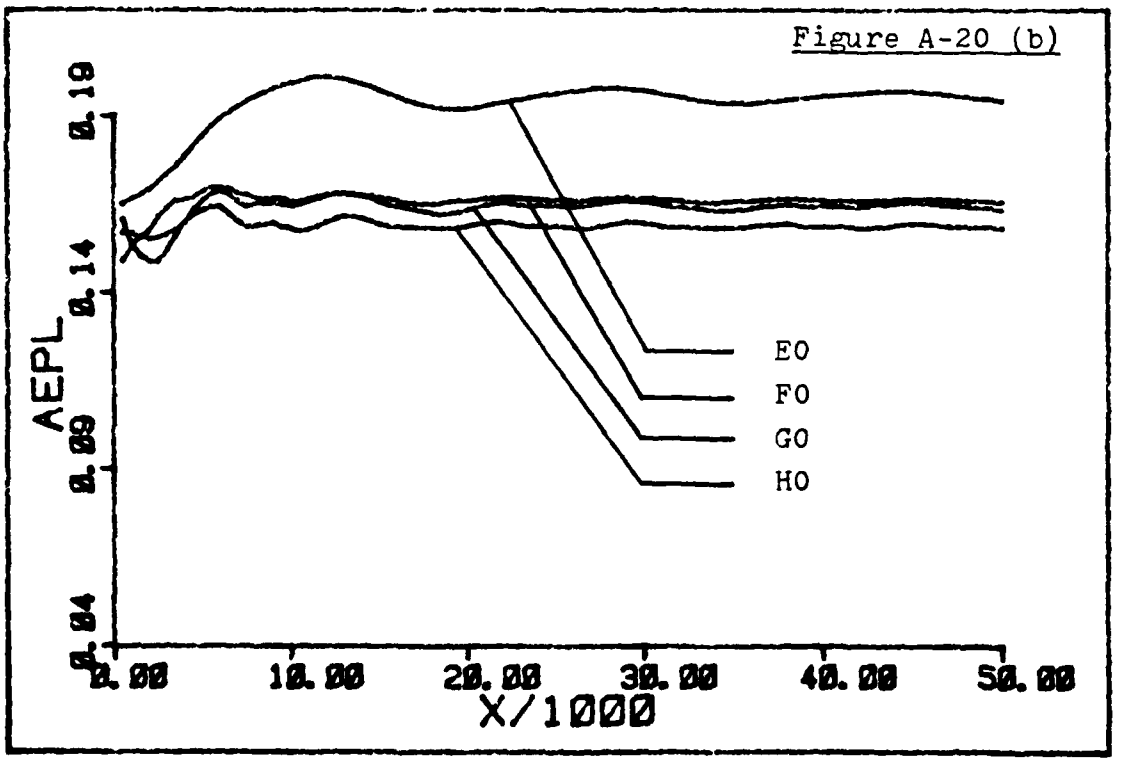


Figure A-21 (a)

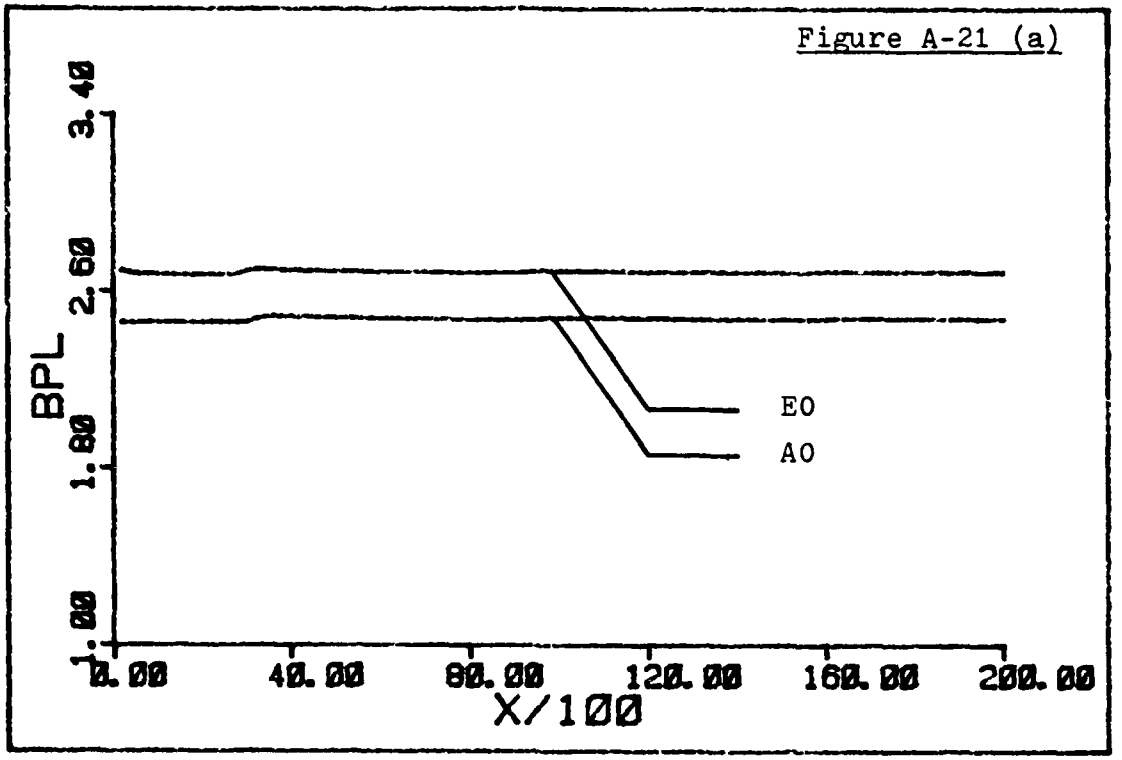


Figure A-21 (b)

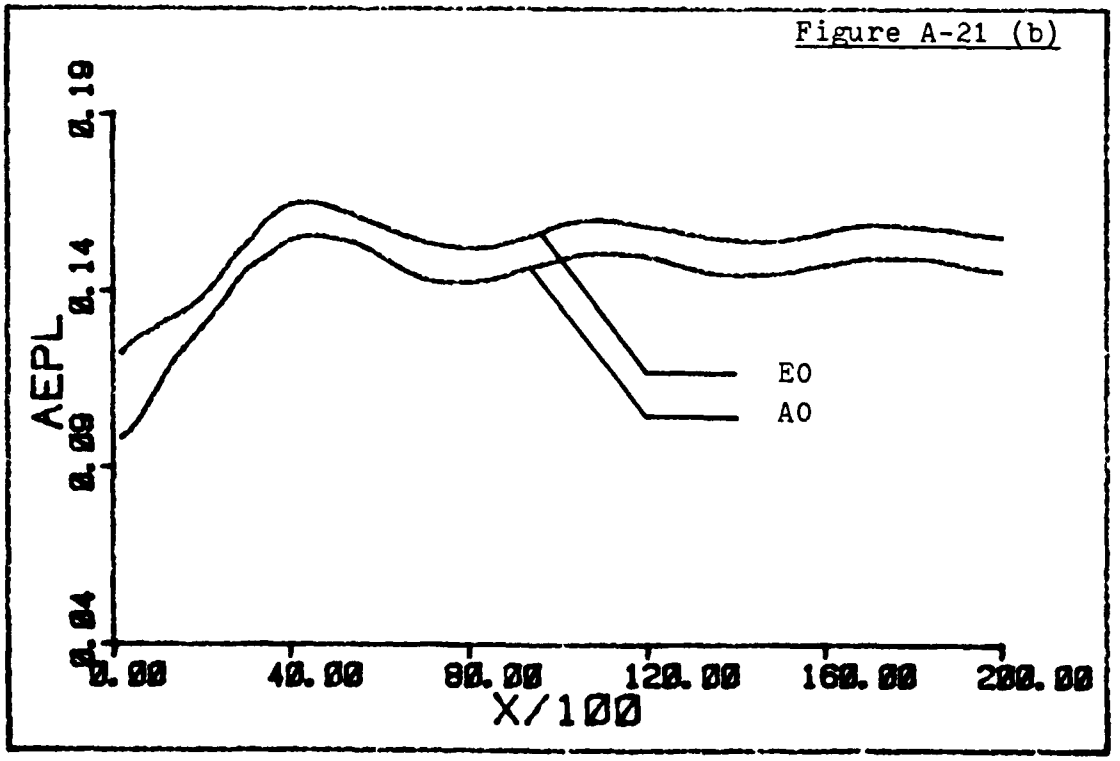


Figure A-22 (a)

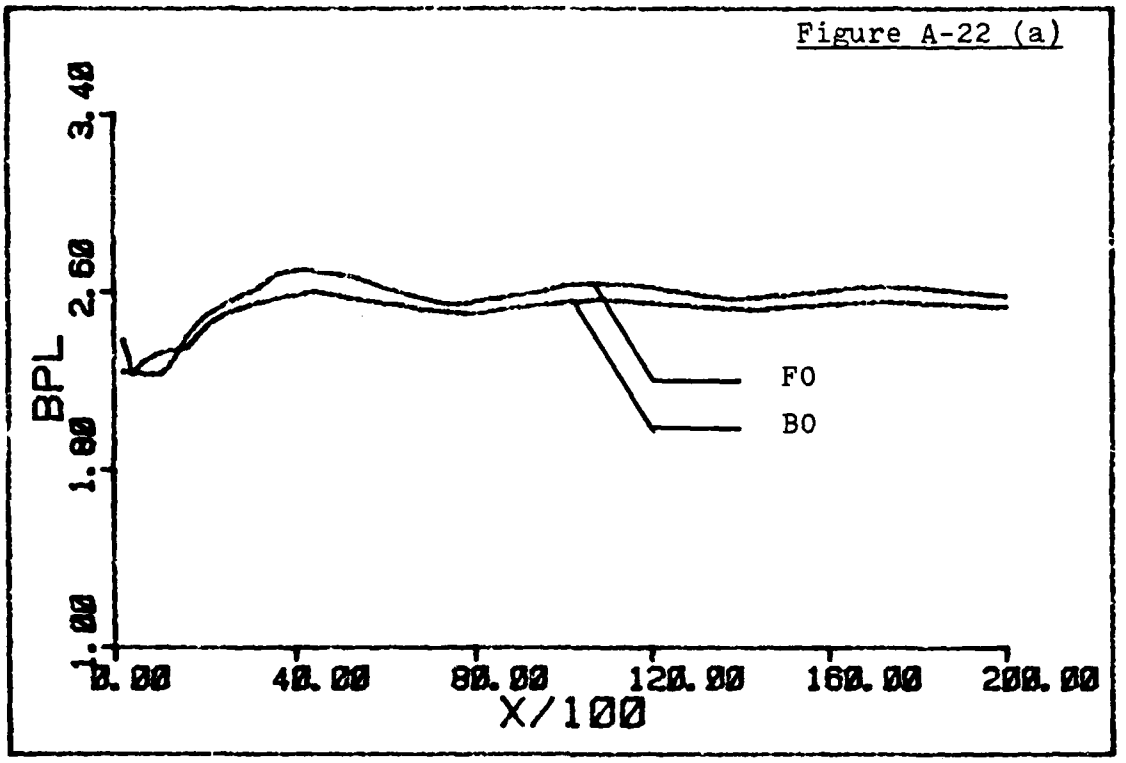


Figure A-22 (b)

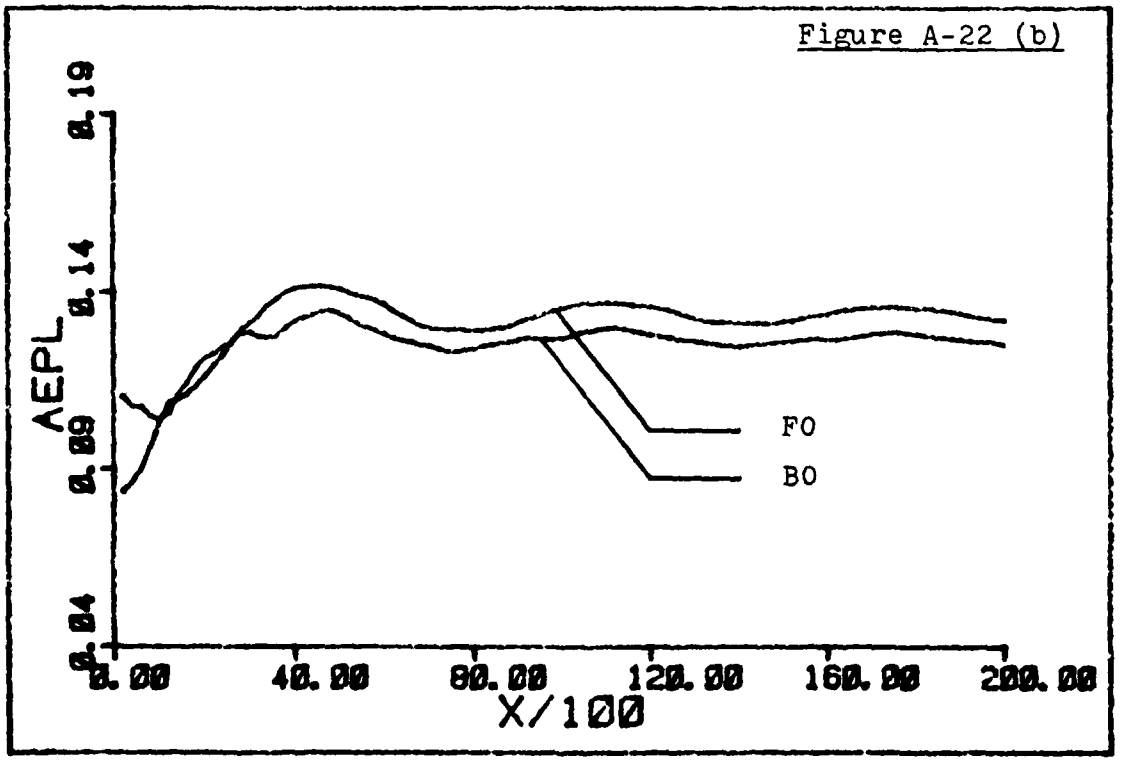


Figure A-23 (a)

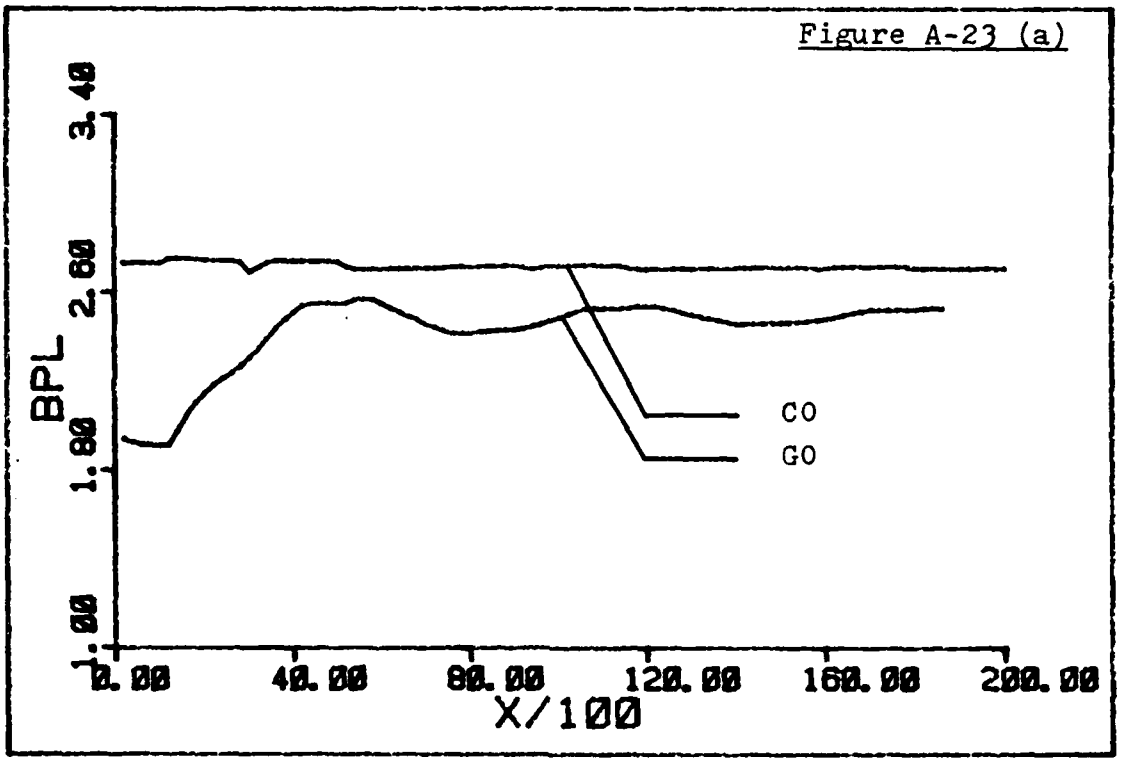


Figure A-23 (b)

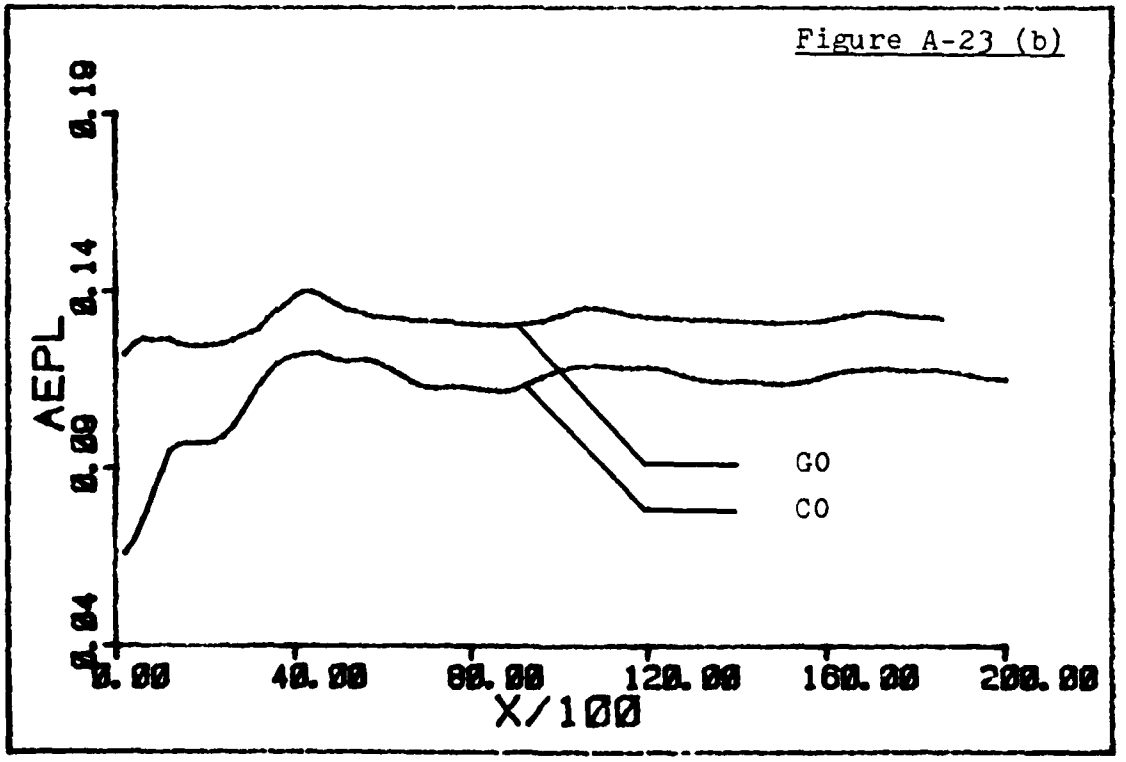


Figure A-24 (a)

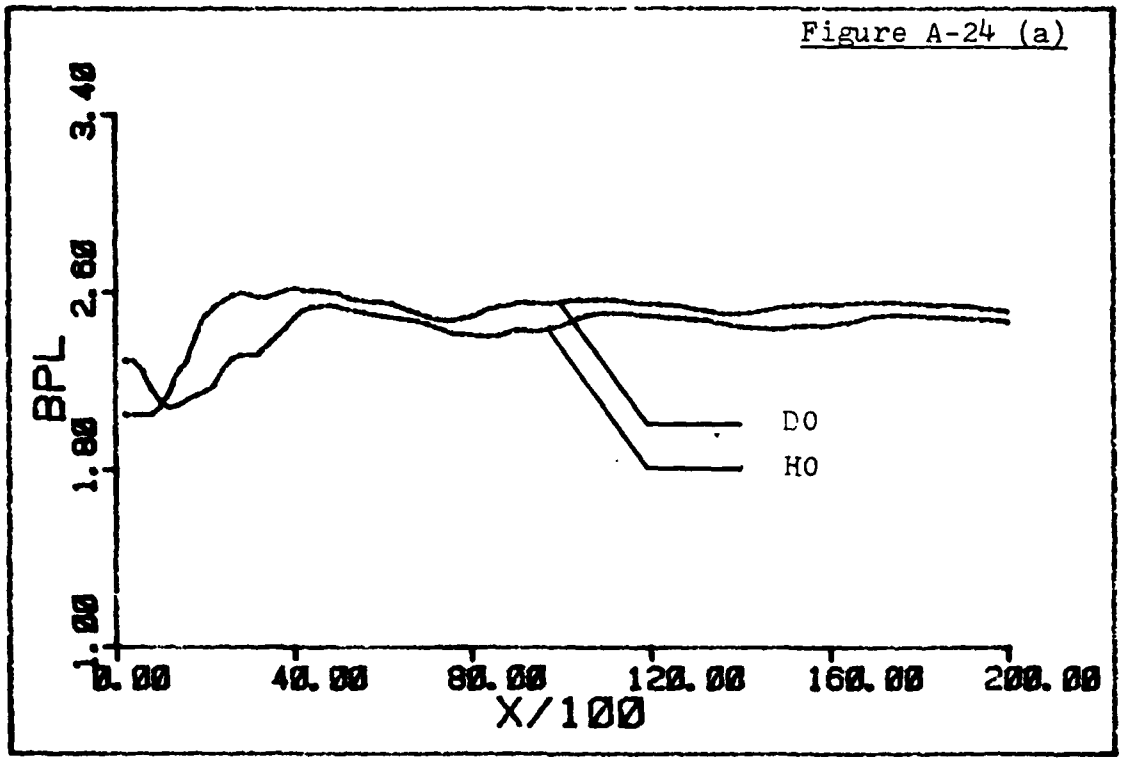
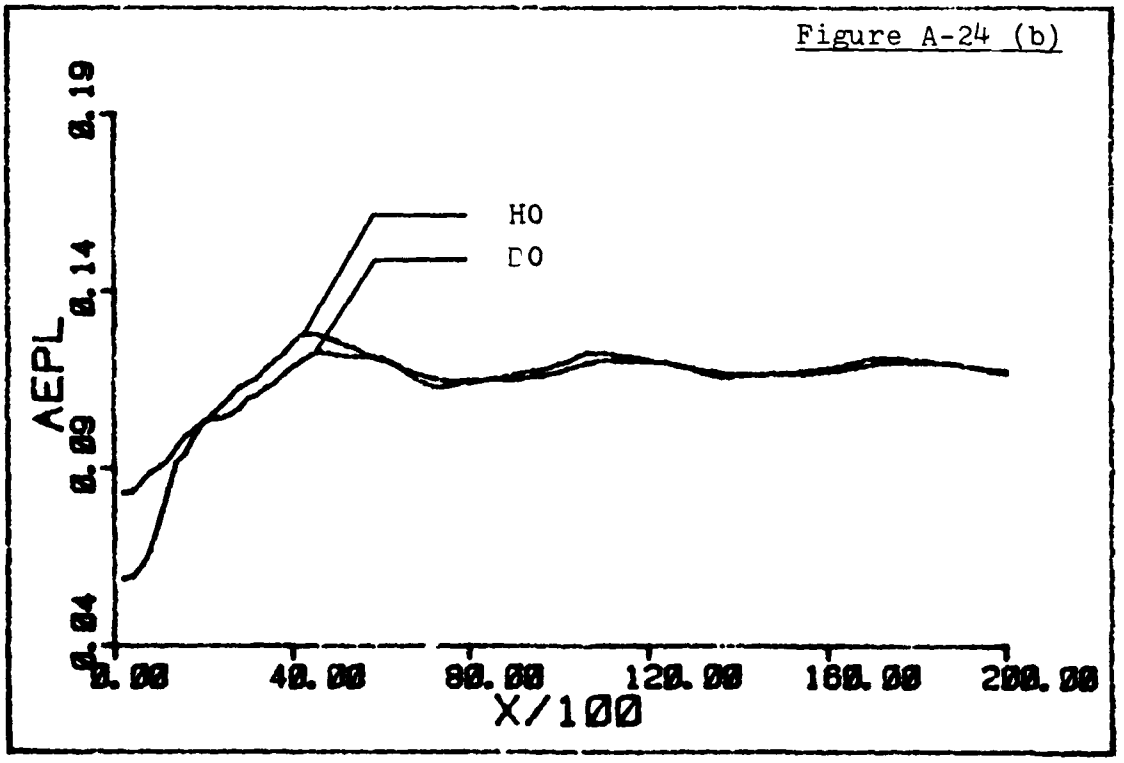


Figure A-24 (b)



Appendix B

Flow Diagrams of the Software

Figure B-1 Overall Program Flow

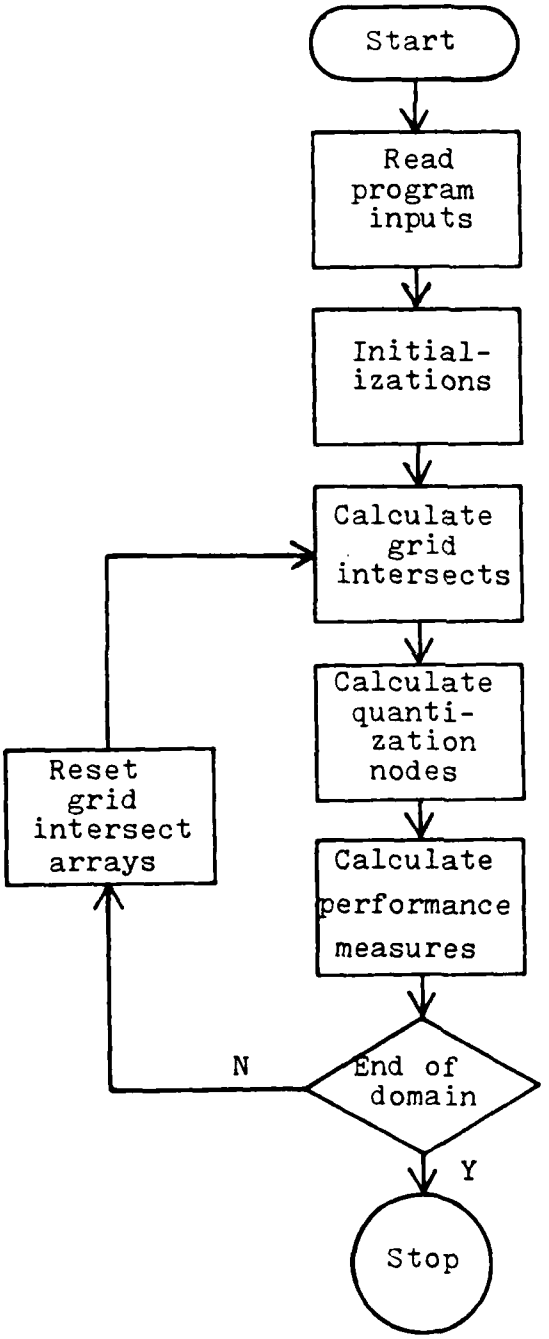


Figure B-2 Calculation of Grid Intersects

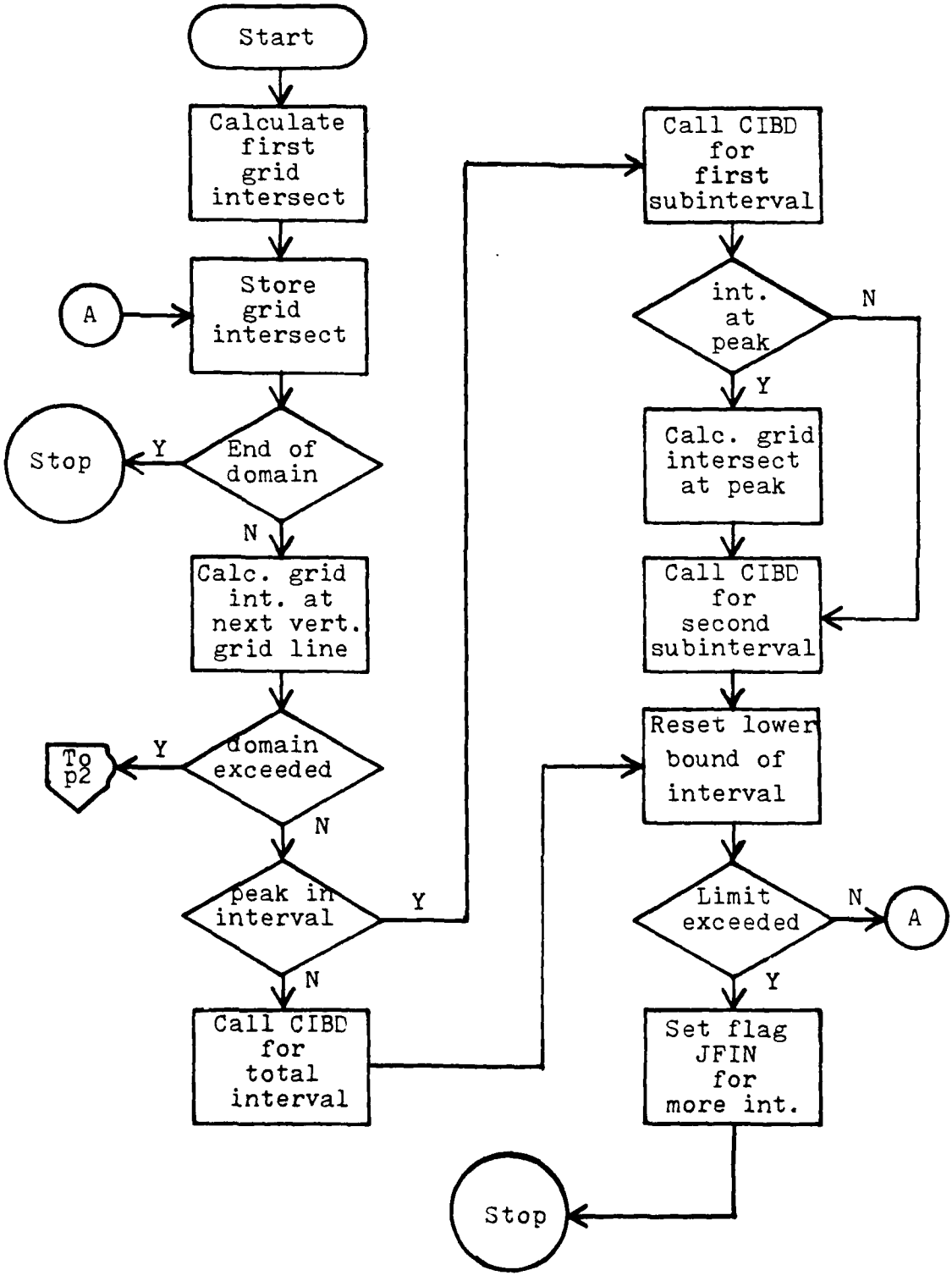


Figure B-2 Page 2

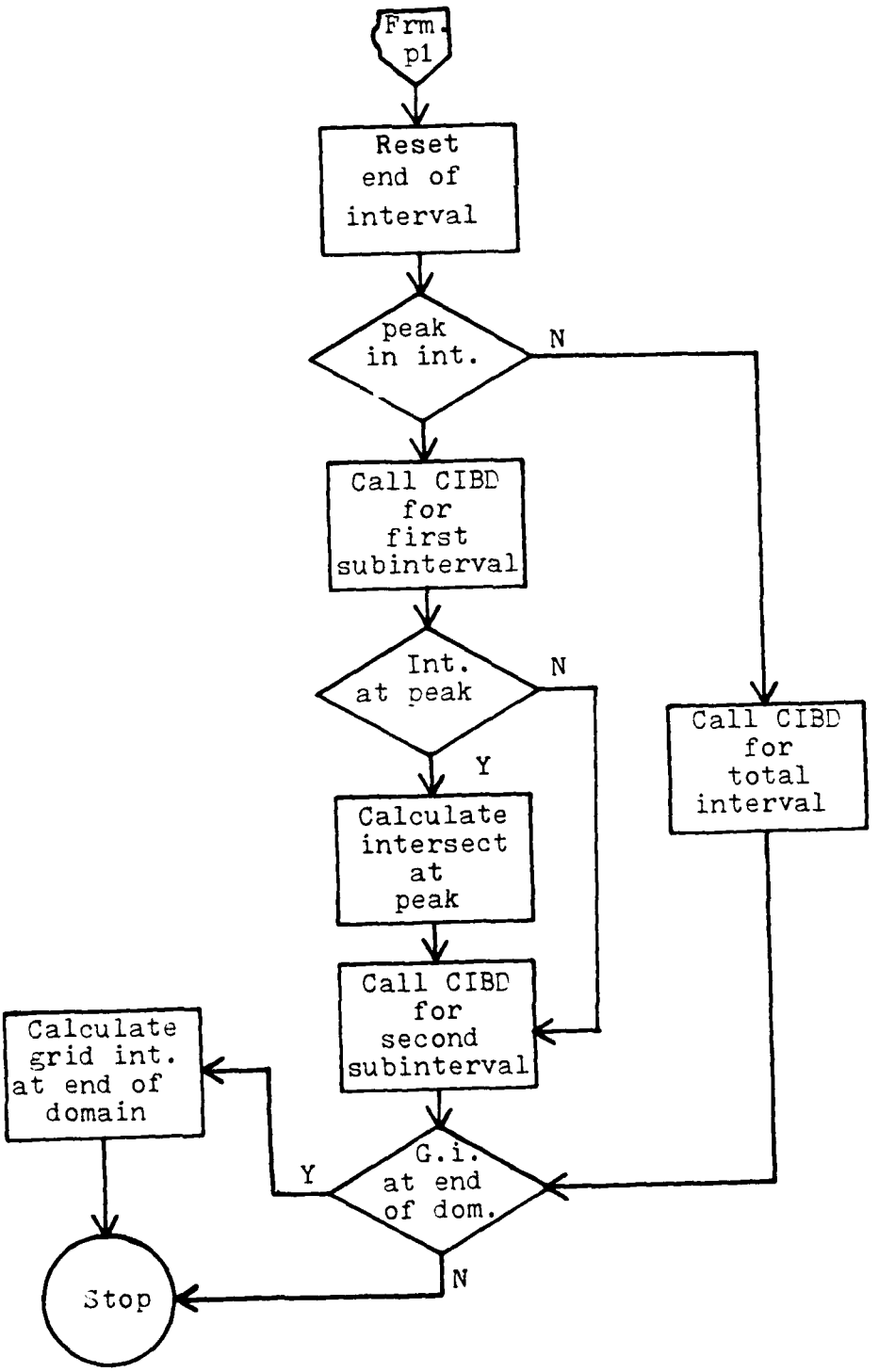


Figure B-3 Calculation of Quantization Nodes

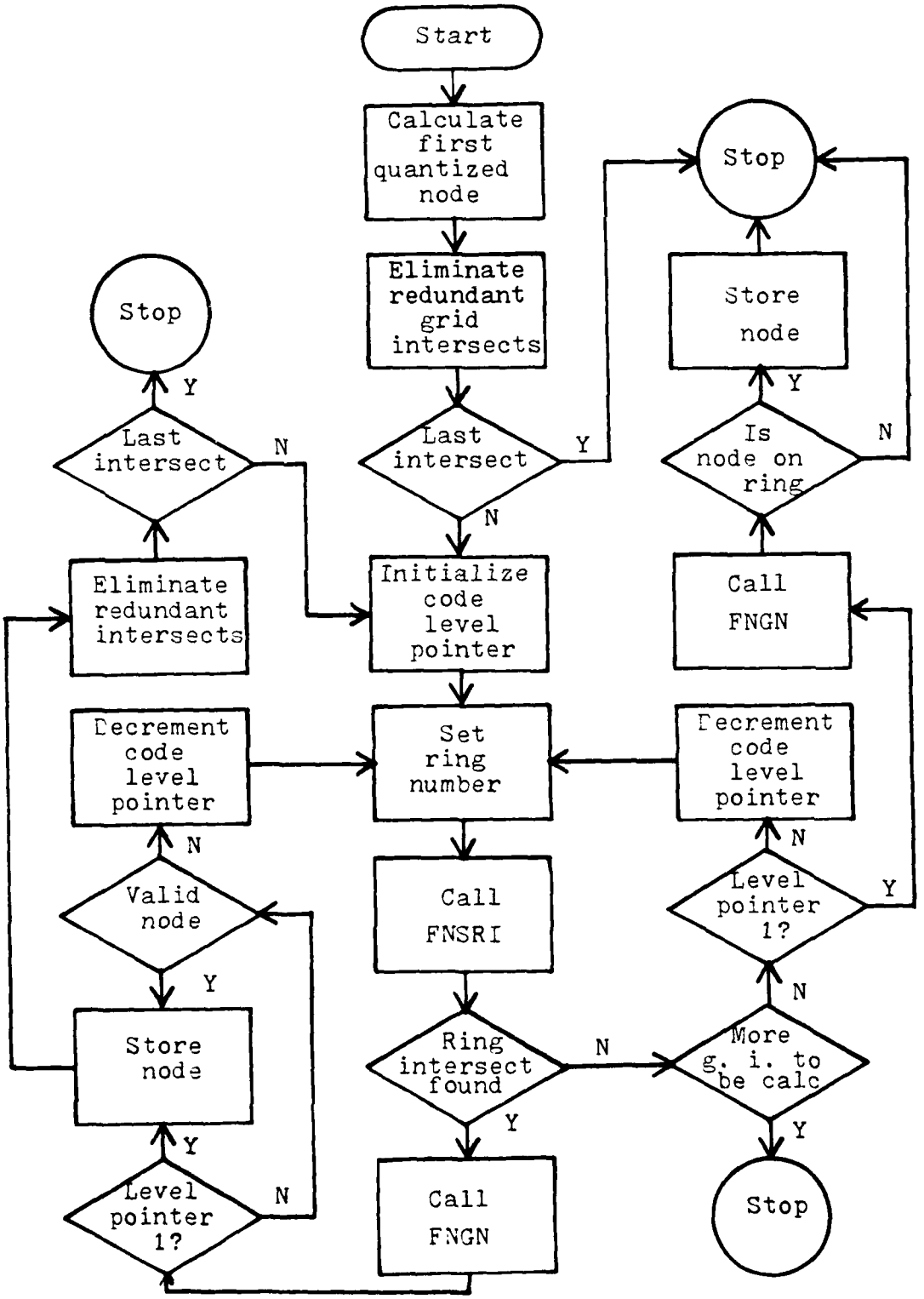
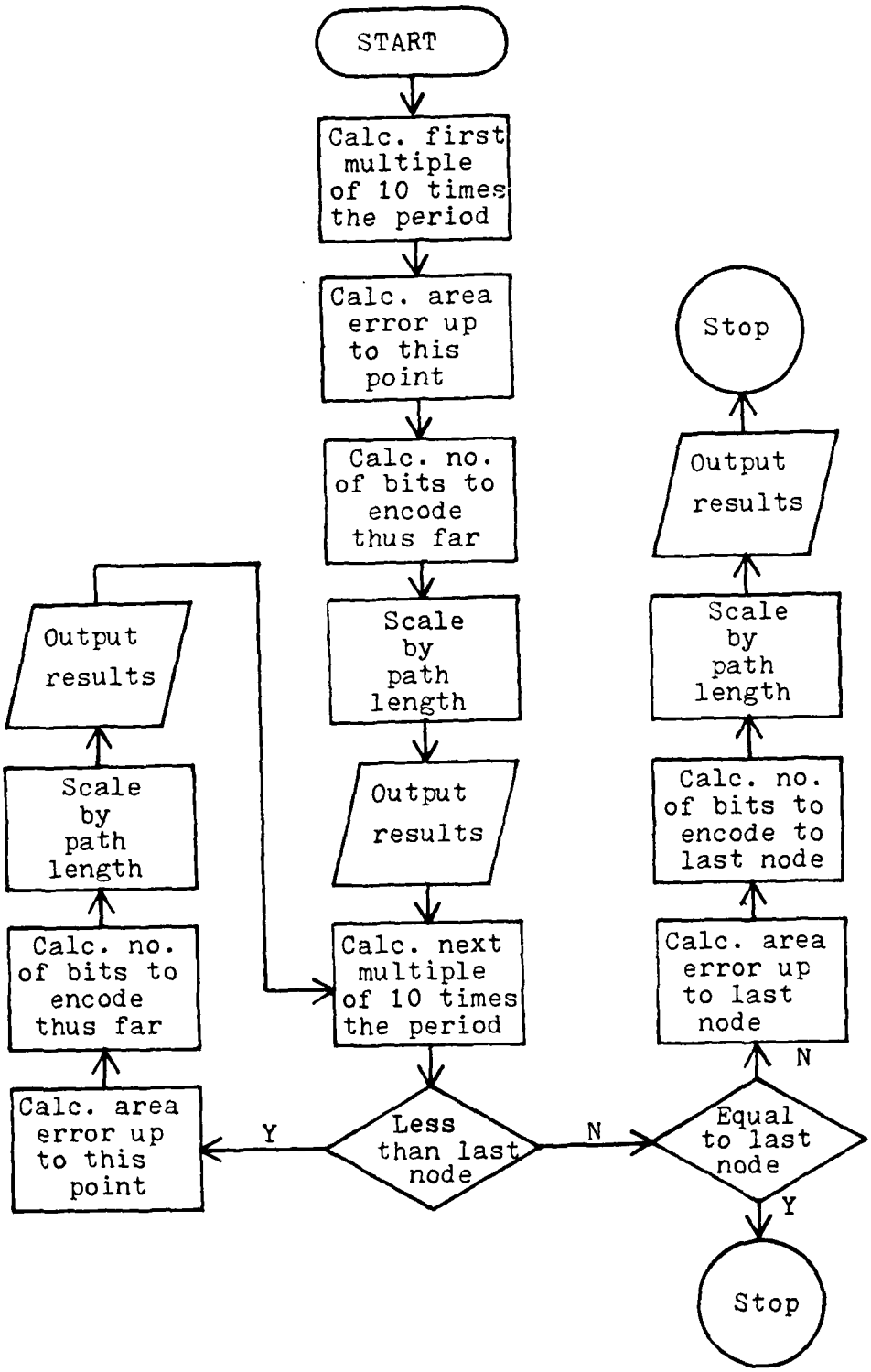


Figure B-4 Calculation of Performance Measures



Vita

Keith Robert Jones was born on 24 March 1958 in Austin, Texas. He graduated from high school in Edmond, Oklahoma in 1976 and attended Oklahoma State University, Stillwater, Oklahoma, from which he received the degree of Bachelor of Science in Electrical Engineering in May of 1981. Upon graduation he received his commission and entered active duty in the Air Force. His first assignment was to the School of Engineering, Air Force Institute of Technology.

Permanent Address: 2633 Edgewood Drive
Edmond, Oklahoma 73034

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/82D-41	2. GOVT ACCESSION NO. AD-A124692	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) GRID-BASED LINE DRAWING QUANTIZATION		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Keith R. Jones 2d. Lt.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio, 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December, 1982
		13. NUMBER OF PAGES 86
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 190-17

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB OH 45433

4 JAN 1983

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Line Drawing Quantization
Generalized Chain Codes
Image Processing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This paper documents a quantitative analysis of the performance of the generalized chain codes when used to quantize specific periodic waveforms. The analysis was performed using a software simulation of the various forms of the code to encode sine waves and circular waves. The performance of the coding schemes was measured in terms of the encoding rate and the area error in the quantization.

Comparisons were made of the performance of the codes for a

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

particular function when the initial phase was altered. Comparisons were also made of the performance of the different forms of the codes used in quantizing the same function. Finally, the performance of the codes for the sine wave was compared with the performance of the same code for a circular wave having the same amplitude and period.